

Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

**Métodos Empíricos para Validação da Reuse
Description Language em Instanciação de Frameworks**

Giovani Salvador

**Dissertação apresentada como
requisito parcial à obtenção do
grau de mestre em Ciência da
Computação**

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

S182m Salvador, Giovani
Métodos empíricos para validação da reuse
description language em instanciação de frameworks /
Giovani Salvador. – Porto Alegre, 2009.
93 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Marcelo Blois Ribeiro

1. Engenharia de Software. 2. Framework.
3. Reutilização de Software. 4. Informática. I. Ribeiro,
Marcelo Blois. II. Título.

CDD 005.1

**Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS**

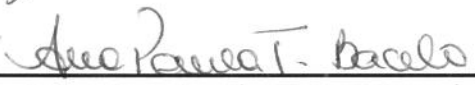



Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO


Dissertação intitulada "**Métodos Empíricos para Validação de Reuse Description Language em Instanciação de Frameworks**", apresentada por Giovani Salvador, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 22/12/08 pela Comissão Examinadora:


Prof. Dr. Marcelo Blois Ribeiro - PPGCC/PUCRS
Orientador


Prof. Dra. Ana Paula Terra Bacelo - PPGCC/PUCRS


Prof. Dr. Joacy Cavalcante de Oliveira - University of Waterloo

Homologada em 15/09/09, conforme Ata No. 16/09 pela Comissão Coordenadora.


Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS

Campus Central

Av. Ipiranga, 6681 - P32 - sala 507 - CEP: 90619-900
Fone: (51) 3320-3611 - Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br

Para minha esposa e filha, que tanto carinho e apoio me passam para enfrentar os desafios diários. Para minha mãe, exemplo de fé e caráter que muito me inspiram.

Agradecimentos

Em primeiro lugar a Deus pai, que sempre me dá forças para superar as barreiras que surgem todos os dias.

À minha esposa e filha, por toda a paciência em entender que o estudo é essencial para o nosso crescimento pessoal e profissional.

À empresa onde trabalho, Dell Computadores, por todo o incentivo ao estudo e à evolução profissional e pessoal de seus funcionários.

Aos professores Marcelo Blois e Toacy Cavalcante pelo apoio, compreensão e orientação, que me instigaram a evoluir enquanto pesquisador.

À Pontifícia Universidade Católica do Rio Grande do Sul por toda a infra estrutura e ferramentas disponibilizadas para a pesquisa.

A todos os funcionários e professores do programa de pós graduação da PUCRS.

Resumo

Toda a instanciação de frameworks segue um processo, seja ele manual ou automatizado. Em um processo manual, um arquiteto ou desenvolvedor segue uma sequência de passos aleatórios com o objetivo de criar uma instância de determinado framework. Em um processo automatizado o arquiteto ou desenvolvedor utiliza uma ou mais ferramentas que sigam passos bem definidos e pré-estabelecidos que irão auxiliar na criação da instância do framework em uso. Verificar o real benefício em usar tais ferramentas é o objetivo deste trabalho, onde dados empíricos serão analisados ao se comparar o processo manual a uma ferramenta de instanciação de frameworks em um experimento. Com isso, pretende-se obter informações que demonstrem, no contexto pesquisado, os ganhos reais de se utilizar uma ferramenta para esse fim ou que as informações demonstrem onde uma ferramenta não agrega valor, permitindo assim sugerir melhorias que justifiquem o uso de tais ferramentas.

Palavras-chave: Reuso. Instanciação. Framework. RDL.

Abstract

Each and every framework instantiation is driven by a process, regardless it is manual or automated. In a manual process, an architect or software engineer follows a sequence of random steps towards creating an instance of a specific framework. In an automated process, the architect or the software engineer takes advantage of one or more tools that follow well-defined, configured steps to guide creation of a framework instance.

To assess the real benefit of such tools is the goal of this study, where empirical data will be analyzed by comparing manual to automated process, using a framework instantiation tool in an experiment. With such data, it is expected to obtain information that shows up, in the context analyzed, the real benefits in adopting a tool to automate instantiation process. With the information gathered is also possible to figure out where those tools do not add any value, allowing suggestions to improve them in order to justify their use.

Keywords: Reuse. Instantiation. Framework. RDL.

Lista de Figuras

Figura 1	Arquitetura framework Struts	23
Figura 2	Classe Utilizando Pontos de Extensão do Struts	24
Figura 3	Arquivo de Configuração do Struts	24
Figura 4	Ciclo de Quality Improvement Paradigm	28
Figura 5	Fases da Experimentação	29
Figura 6	Relacionamento entre Objetivos, Questões e Métricas em GQM	33
Figura 7	Atores e atividades do processo de instanciação	44
Figura 8	A ferramenta Reuse Tool executando scripts RDL sobre um modelo XMI	45
Figura 9	Modelo final gerado e estendido	45
Figura 10	Problema com interfaces na RDL	47
Figura 11	Problema com parâmetros de operações na RDL	47
Figura 12	RDL com suporte a interfaces	48
Figura 13	Diagrama de Classes do Framework MySubmission.	58
Figura 14	Classes de persistência.	59
Figura 15	Classe SubmissionFacade	60
Figura 16	Gráfico de Boxplot para a Variável Erros	66
Figura 17	Gráfico de Boxplot para a Variável Quantidade de Acessos	67
Figura 18	Gráfico de Boxplot para a Variável Tempo	67
Figura 19	Diagrama de Classes Framework MySubmission	84

Lista de Tabelas

Tabela 1	Falácias e refutações sobre experimentação em ES	37
Tabela 2	Problemas e soluções para Replicação com Comunicação Zero	39
Tabela 3	Sumário dos problemas na execução do experimento com Comunicação Ágil	40
Tabela 4	Sumário dos problemas na execução do experimento com Comunicação Ágil	41
Tabela 5	Relação entre questão, métrica e forma de coleta.	52
Tabela 6	Distribuição dos participantes	61
Tabela 7	Tempos de Execução dos Procedimentos	63
Tabela 8	Teste de Shapiro-Wilk para as Variáveis do Experimento	65
Tabela 9	Teste de Levene para Igualdade das Variâncias	66
Tabela 10	Teste T	68

Lista de Siglas

RDL	Reuse Description Language	22
MVC	Model-View-Controller	23
ES	Engenharia de Software	32
XMI	XML Metadata Interchange	43
UML	Unified Modeling Language	44
GQM	Goal-Question-Metric	51

Sumário

1	Introdução	21
1.1	Motivação	21
1.2	Contextualização do Problema	22
1.3	Estrutura do trabalho	25
2	Engenharia de Software Empírica	27
2.1	Experimentação em Engenharia de Software	27
2.1.1	Terminologia Relacionada a Experimentos	30
2.1.2	Variáveis de Resposta em Engenharia de Software	32
2.1.3	Experimentação em Ambientes Reais	33
2.2	Porque não são Realizados Experimentos em Engenharia de Software?	35
2.3	Problemas e Considerações quanto a Replicação de Experimentos em Engenharia de Software	37
2.3.1	Replicação com Comunicação Zero	38
2.3.2	Replicação com Comunicação Ágil	39
2.3.3	Exercitando com Comunicação Ágil	41
3	RDL - Reuse Description Language	43
3.1	A Ferramenta Reuse Tool	44
3.2	Limitações em RDL	46
3.2.1	Falta de Suporte a Interfaces	46
3.2.2	Falta de Suporte a Parâmetros em Operações	47
3.2.3	Conclusão das Limitações	48
3.3	Melhorando RDL	48
3.3.1	Suporte a Interfaces	48
3.3.2	Suporte a Parâmetros em Operações	49
4	O Estudo Experimental	51
4.1	Objetivo e Hipóteses do Experimento	51
4.1.1	Objetivo do Estudo Experimental	51
4.1.2	Objetivo da Medição	51
4.2	Planejamento do Experimento	55
4.2.1	Caracterização do Contexto	55
4.2.2	Experimento Piloto	56
4.2.3	Instrumentação	56
4.3	Definição do Framework	57
4.3.1	MySubmission Framework	58
4.4	Execução do Experimento	60

4.4.1	Participantes	60
4.4.2	Procedimentos de Execução do Experimento	61
4.4.3	Coleta dos Dados	63
4.4.4	Análise dos dados	65
4.4.5	Análise Qualitativa	68
5	Conclusão	71
5.0.6	Trabalhos Futuros	72
	Referências	73
	Apêndice A – Questionário para entrevista	75
	Apêndice B – Caso de Uso para o Experimento	77
	Apêndice C – Script RDL Utilizado no Experimento	79
	Apêndice D – Documentação do Framework	81
	D.1 Introdução	81
	D.2 Uso do Framework	81
	D.2.1 Como obter referências de objetos para validações, persistência e notificações	81
	D.2.2 Como implementar validações (Regras de Negócio)	82
	D.2.3 Persistência de Dados	82
	D.2.4 Notificações	83
	D.2.5 Submissão de Artigos	83
	D.2.6 Diagrama de Classes do Framework	84
	Apêndice E – Arquivo de Especificação da Reuse Description Language	85

1 Introdução

1.1 Motivação

O reuso é uma das técnicas de engenharia de software utilizada para promover desenvolvimento de software de uma forma mais rápida, com mais qualidade e com menos erros [1]. Mais rápida pois não se está desenvolvendo um pedaço de código desde o início e sim reutilizando algo existente. Mais qualidade pois funções e rotinas usadas em mais de um lugar já sofreram modificações decorrentes de erros encontrados ou ajustes em usos iniciais sendo, assim, corrigidas e liberadas para que sejam utilizadas novamente. Menos erros pois o uso decorrente de funções e rotinas genéricas mostra situações das mais variadas dentro dos sistemas e que também contribuem para que trechos reutilizáveis sejam postos à prova, adaptados para as mais variadas situações ou corrigidos quando erros são encontrados. As rotinas e/ou funções genéricas que promovem o reuso podem ser encontrados das mais diversas formas, dentre elas:

- Bibliotecas de Código: Trechos de código compilados e empacotados de uma forma a serem incorporados nos sistemas que fazem uso de suas rotinas.
- Código Fonte: Trechos de código não compilados cujos códigos fonte estão disponíveis de uma forma a serem colocados no fonte dos sistemas que os utilizarão.
- Frameworks: Contém um ou mais bibliotecas de código que juntas provém suporte a uma ou mais funcionalidades tais como segurança de aplicações, sistemas para internet ou persistência de dados, entre outras.
- Design Patterns [2]: Soluções documentadas para problemas recorrentes, geralmente encontrados na maioria das aplicações.

Entretanto, a tarefa de reutilizar um ou mais artefatos para compor um sistema pode não ser tão simples pois depende de uma série de fatores que aumentam ou diminuem a curva de aprendizado num determinado artefato. Um fator é a quantidade e/ou qualidade da documentação de determinada biblioteca, framework ou componente. A qualidade da documentação de determinado artefato reutilizável não é algo que se consegue medir de forma quantitativa, ou seja, depende de aspectos subjetivos que avaliam se quantidade e qualidade são suficientes para a documentação do artefato em questão.

Uma vez que um desenvolvedor tenha em mãos um artefato reutilizável o passo seguinte é promover a sua inserção dentro do código fonte de um sistema, no caso de bibliotecas de código ou criar um sistema com base nesse artefato, como é o caso de frameworks. Esse passo é chamado de instanciação.

No caso de frameworks, toda instanciação segue um processo, seja ele manual ou automatizado, de criação de um sistema que utilize o que o framework pode prover em termos de tarefas prontas. Por processo manual, entende-se sem o uso específico de uma ferramenta que auxilie e guie arquitetos de software e desenvolvedores na instanciação. O desenvolvedor ou arquiteto, de posse do framework e de sua documentação, começa a utilizar os pontos de extensão, com base no que a documentação descreve ou com base em exemplos providos juntos com o framework.

Por automatizado, entende-se um processo que já utilize alguma ferramenta responsável por guiar arquitetos e desenvolvedores numa correta seqüência de passos para atingir o objetivo final, que é fazer uso dos pontos de variabilidade [1] de determinado framework para gerar uma aplicação final resultante do processo de instanciação.

Conforme a literatura relacionada, ferramentas que automatizam o processo de instanciação de frameworks podem prover uma série de benefícios tais como:

- Guiar arquitetos e desenvolvedores numa correta seqüência de instanciação.
- Interação com o arquiteto/desenvolvedor para o processo de instanciação.
- Redução da curva de aprendizado para um determinado framework.

Como conseqüência desses benefícios pode-se encontrar uma redução no número de erros encontrados durante o processo de instanciação bem como um aumento na produtividade. Entretanto, o desafio para esse trabalho é justamente encontrar evidências de que as ferramentas de instanciação de frameworks (o foco desse trabalho é em RDL [1]) realmente atingem esses benefícios e que o ganho real medido do uso de uma ferramenta de auxílio à instanciação é significativo se comparado a um processo manual.

1.2 Contextualização do Problema

Para contextualizar o problema, imaginemos o uso de um dos mais famosos frameworks para desenvolvimento de aplicações web utilizando Java, o framework Struts [3]. Esse framework auxilia desenvolvedores e arquitetos a modelarem aplicações para web sem se preocuparem com detalhes específicos de protocolos como HTTP (HyperText Transfer Protocol) ou FTP (File Transfer Protocol). O framework Struts foi criado em 2000 e ajudou a popularizar a

linguagem Java para o desenvolvimento de soluções para a internet. Struts se baseia no padrão de projetos MVC [4] ou Model-View-Controller, que busca separar o modelo (Model) da interface ou apresentação (View) e dos controladores (Controller). O modelo representa os dados ou a lógica de negócio. A apresentação representa a interface, as páginas apresentadas para o usuário. E, por fim, o controlador junta o modelo com a apresentação, controlando a navegação e a ligação entre as duas camadas. Com a separação em camadas sugerida por MVC chega-se a uma melhor manutenibilidade das aplicações desenvolvidas além de uma separação maior de especialidades uma vez que cada camada cuida de uma função diferente.

A Figura 1 dá uma visão da arquitetura do Framework Struts.

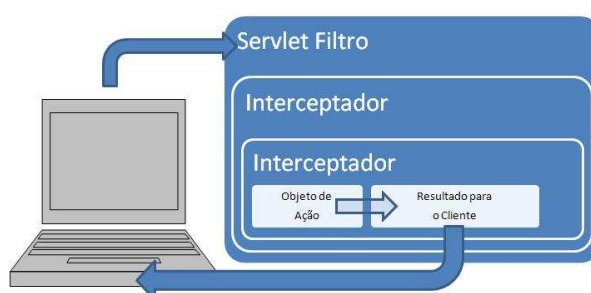


Figura 1 – Arquitetura framework Struts

[3]

Basicamente o framework intercepta requisições HTTP através de seus interceptadores (Interceptors) e as repassa para o componente responsável por tratar essa requisição (classes Action) e transformá-la em um resultado que é tipicamente apresentado para um usuário sob a forma de uma página web.

Embora o framework Struts tenha se tornado bastante popular, desenvolvedores (principalmente os iniciantes na tecnologia) precisam do auxílio da documentação para entender os pontos de variabilidade existentes no framework para uma correta criação das classes que farão uso desses pontos além dos arquivos de configuração necessários para mapear os elementos que farão parte de cada camada do MVC. Além disso, o apoio de um ambiente de desenvolvimento é fundamental para uma correta criação desses artefatos. Todo esse processo de aprendizado resulta em tempo e uma série de erros podem ocorrer no decorrer desse processo até que se atinja o produto final desejado. Embora o aprendizado fique para próximos trabalhos utilizando o mesmo framework, novos ou os mesmo erros podem ocorrer novamente, resultando em tempo despendido para se atingir o produto e reprodução dos mesmos problemas.

Por exemplo, a Figura 2 mostra uma classe que utiliza dois pontos de extensão do framework Struts. Essa classe implementa uma interface (identificado por *implements Preparable*) e estende outra classe (identificado por *extends ActionSupport*).

```

...
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.Preparable;
...

public class EmployeeAction extends ActionSupport implements Preparable {
    ...
}

```

Figura 2 – Classe Utilizando Pontos de Extensão do Struts

[3]

Ainda, o framework Struts requer configurações adicionais como mostra a Figura 3.

```

<struts>
  <!-- Configuration for the default package. -->
  <package name="default" extends="struts-default">

    <!-- Default interceptor stack. -->
    <default-interceptor-ref name="paramsPrepareParamsStack"/>

    <action name="index" class="com.aurifa.struts2.tutorial.action.EmployeeAction" method="list">
      <result name="success" />WEB-INF/jsp/employees.jsp</result>
      <!-- we don't need the full stack here -->
      <interceptor-ref name="basicStack"/>
    </action>

    <action name="crud" class="com.aurifa.struts2.tutorial.action.EmployeeAction" method="input">
      <result name="success" type="redirect-action">index</result>
      <result name="input" />WEB-INF/jsp/employeeForm.jsp</result>
      <result name="error" />WEB-INF/jsp/error.jsp</result>
    </action>
  </package>
</struts>

```

Figura 3 – Arquivo de Configuração do Struts

[3]

Nesse pequeno exemplo de uso do framework Struts é possível perceber que um desenvolvedor precisaria conhecer os detalhes do mesmo para saber que necessita estender uma classe, implementar uma interface e ainda colocar informações em arquivo de configuração. Nesse momento, algumas perguntas são naturalmente feitas a respeito do uso do framework, tais como:

- É sempre necessário implementar uma interface?
- Em que situações se precisa estender *ActionSupport* (no caso do framework Struts)?
- Que informações são colocadas no arquivo de configuração do framework e quais as opções disponíveis?

Enfim, existe toda uma curva de aprendizado relacionada ao uso do framework e ao uso de seus pontos de variabilidade. Em que nível de detalhe engenheiros de software precisam conhecer os detalhes de um framework? E se existisse alguma ferramenta que os guiassem no correto uso do mesmo, ajudaria? Faria alguma diferença em relação ao tempo de desenvolvimento, acessos necessários à documentação ou até mesmo à diminuição de erros decorrentes do uso do framework?

Aqui cabe observar que o Struts não é o objeto de estudos desse trabalho, foi utilizado apenas como exemplo de framework para contextualizar um problema, que é o do conhecimento de detalhes de um framework para criação de aplicações.

1.3 Estrutura do trabalho

Esta dissertação está dividida da seguinte maneira:

- O Capítulo 1 fez a introdução a esse trabalho bem como a motivação por trás do mesmo.
- O Capítulo 2 apresenta a fundamentação teórica a respeito da Engenharia de Software Empírica.
- O Capítulo 3 apresenta a Reuse Description Language, suas limitações e melhorias efetuadas por esse trabalho.
- O Capítulo 4 mostra como foi realizado o Estudo Experimental com a RDL, o planejamento, a construção de uma framework para o experimento, a análise dos dados e demais considerações.
- Por fim, o Capítulo 5 faz a conclusão desse trabalho e discorre sobre trabalhos futuros.

2 Engenharia de Software Empírica

Empirismo é considerado um sistema filosófico que atribui à experiência a origem do conhecimento humano, ou seja, conhecimento adquirido pela experiência, não pelo estudo [5]. Empírico é algo ou alguém que se guia pelas evidências geradas pela experiência. O quanto a engenharia de software tem usado da experiência para propor novos modelos e práticas para uma melhor construção de aplicativos de software ou para melhorar os processos existentes? Como será visto nesse capítulo, outras áreas já usam a experimentação (base do empirismo) para a produção de novos produtos ou a melhoria de produtos e/ou modelos existentes. Essa seção tem como objetivo apresentar a importância da aplicação de experimentos para construir conhecimento baseado em evidências.

Embora manufatura e medicina sejam áreas bem mais antigas que a Engenharia de Software (ES) e, portanto, têm evoluído mais ao longo do tempo tanto em técnicas como em resultados, seria possível a ES fazer uso das boas práticas adotadas por essas e outras áreas que têm utilizado a experimentação como prática para melhorar processos ou criar novos produtos? O que a ES difere de outras áreas que precisa ser levado em consideração na hora de se fazer um experimento? A seção seguinte discute motivos relacionados com o baixo uso de experimentos em ES. Após, discute-se o uso de experimentação na ES, abordando noções e conceitos da área. No fim desse capítulo, alguns problemas relacionados à experimentação são descritos para que sejam devidamente evitados.

2.1 Experimentação em Engenharia de Software

Existem duas abordagens [6] para executar investigações empíricas: Pesquisa Quantitativa e Pesquisa Qualitativa. Pesquisa quantitativa tem como objetivo obter números provenientes do objeto de estudo e suas variáveis, como por exemplo, o percentual de aumento de produtividade no desenvolvimento de um software ao se adicionar templates de código ao ambiente de desenvolvimento em comparação ao mesmo software sendo desenvolvido sem o recurso citado. Nesse caso, o percentual obtido fornece o que a pesquisa quantitativa se propõe a fazer, fatos em forma de números para evidenciar o que antes era uma idéia. Já a Pesquisa Qualitativa não foca somente em números mas também em explicar fatores de qualidade associados ao dia-a-dia de uso de alguma técnica, ferramenta e/ou processo. Por exemplo, a Pesquisa Qualitativa serve para demonstrar por que os desenvolvedores tem mais produtividade com a linguagem de

programação A do que com a linguagem de programação B. Nesse caso, é sabido que a linguagem A é melhor que a B e que os desenvolvedores são mais produtivos utilizando A. Como resultado da Pesquisa Qualitativa tem-se explicações em texto, gráficos e, em algumas vezes, também números.

Para chegar a esses resultados, é preciso seguir alguns passos na condução da experimentação. Embora a literatura não seja abrangente [7] no que diz respeito à resultados e exemplos de experimentação em ES e também em relação a metodologias, existem algumas técnicas, fases e passos a serem seguidos e aplicados para conduzir experimentos em ES. Victor Basili [8] definiu as fases de experimentação como sendo a **definição** do projeto de um experimento, a **observação** (quando da aplicação do experimento), a **coleta e análise de dados** e a **validação** do produto que está sendo testado. Para suportar essas fases [2] usou um paradigma experimental chamado *Quality Improvement Paradigm* (QIP - [9]). QIP é um modelo iterativo que suporta a melhoria de processos de desenvolvimento de software de uma organização. É uma maneira de se desenvolver práticas dentro das organizações, experimentando-as, capturando-as e empacotando-as para que se possa reutilizar no futuro. QIP surgiu de um trabalho realizado dentro do Departamento de Engenharia de Software da NASA [8]. A Figura 2.1 dá uma visão geral do ciclo de melhoria sugerido no paradigma QIP [10].



Figura 4 – Ciclo de Quality Improvement Paradigm

[10]

Os passos do QIP são:

- Caracterizar e projetar o ambiente.

- Estabelecer os objetivos para uma bem sucedida melhoria de projeto (aplicados a ferramentas, processos, etc.) e posterior aprendizado da organização.
- Escolher adequadamente os processos e ferramentas para o projeto em questão e para o estudo.
- Executar os processos, construir o produto em questão, coletar e validar dados com base nos objetivos e analisar os dados para prover *feedback* imediato para uma possível ação corretiva.
- Analisar os dados para validar as práticas, determinar problemas e fazer recomendações para futuras melhorias de processos.
- Finalmente, empacotar a experiência na forma de modelos estruturados que sirvam de conhecimento para projetos futuros.

Victor R. Basili [8] criou o conceito de Fábrica de Experiências, que é uma estrutura organizacional que suporta o QIP. O objetivo da Fábrica de Experiências é representar uma forma de ambiente de laboratório para o desenvolvimento de software, onde projetos são desenvolvidos e estudados e onde modelos podem ser criados e/ou aprimorados.

Já Natalia Juristo [7] definiu as fases da experimentação como sendo **definição** dos objetivos da experimentação, **desenho/projeto** nos experimentos, **execução** dos experimentos e, por fim, **análise** dos dados e resultados coletados dos experimentos. A Figura 5 demonstra a relação entre as fases.



Figura 5 – Fases da Experimentação

[7]

Segundo essa abordagem, as responsabilidades de cada fase são dispostas como descrito a seguir: Na definição dos objetivos uma hipótese geral é transformada em termos de quais variáveis do objeto de estudo (ou fenômeno a ser estudado) serão testadas. Como exemplo, suponha um experimento que irá analisar quão eficientes são duas técnicas de teste para encontrar determinado tipo de erro em aplicativos de software. Uma possível hipótese para este exemplo seria: "A técnica A consegue encontrar mais erros do tipo 1 do que a técnica B". Essa é uma hipótese quantificável uma vez que pode ser medido (número de erros detectados por cada técnica).

Já a fase de desenho/projeto do experimento envolve planejar sob que condições o experimento será executado, definindo quais as variáveis que irão afetar o experimento. No exemplo anterior, as variáveis são as duas técnicas de detecção de erros. Nessa fase também se discute quem irá participar do experimento, quais as condições de ambiente e quantas vezes o experimento será executado. A fase de execução envolve executar o experimento seguindo tudo que foi previamente planejado. Uma vez que o experimento tenha sido executado, então é momento de começar a analisar (fase de análise) os dados coletados.

2.1.1 Terminologia Relacionada a Experimentos

Dentro de cada fase da experimentação um ou mais conceitos são estudados ou aplicados. Esses conceitos têm uma terminologia embora a literatura não os trate de forma universal ao longo das diferentes propostas de experimentação em ES. Como base desse trabalho, utilizou-se a terminologia apresentada em [7]:

- **Unidade Experimental (*Experimental Unit*):** são os objetos sobre os quais o experimento é executado. Se um determinado experimento propõe comparar duas abordagens para instanciação de frameworks [1] então a unidade experimental é a fase de instanciação de frameworks.
- **Sujeitos Experimentais (*Experimental Subject*):** são as pessoas responsáveis por aplicar o que se está experimentando. No caso do exemplo de instanciação de frameworks, os sujeitos experimentais são os desenvolvedores que irão receber um framework e precisarão instanciá-lo usando as técnicas que estão sendo estudadas/comparadas. É importante salientar que em ES, diferentemente de outras áreas/disciplinas, o fator emocional dos sujeitos precisa ser levado em consideração. Ainda, a seleção de quais sujeitos farão parte do experimento pode levar em consideração o grau de experiência ou maturidade do sujeito com relação às técnicas sendo aplicadas.
- **Variáveis de Resposta (*Response Variables*):** um ou mais resultados quantitativos, provenientes da execução de um experimento, formam as variáveis de resposta, também chamadas de variáveis dependentes. Em outras palavras, representam a saída da execução de um experimento. Adiante será vista uma abordagem para medir variáveis de resposta. No exemplo de técnicas/ferramentas de instanciação de frameworks, uma variável de resposta poderia ser o tempo despendido para estender um framework usando diferentes técnicas. Aqui vale observar que cada valor de uma variável de resposta deve ser analisado e, com base nessa análise, pode-se inferir se uma ou mais hipóteses podem ser validadas.
- **Parâmetros:** os parâmetros são todas as características (sejam quantitativas ou qualitativas) que não irão se modificar ao longo da execução de um experimento e que, supõe-se,

não modificarão o seu resultado. No exemplo de instanciação de frameworks, a experiência da equipe é um exemplo de parâmetro quando não se deseja se não se deseja avaliar a influência da mesma nos resultados.

- Fatores ou variações provocadas (*Factors*): diferentemente dos parâmetros, os fatores são características que influenciam as variáveis de resposta, ou seja, intencionalmente variam durante o experimento e afetam os resultados. No exemplo de instanciação de frameworks, os fatores são as técnicas de instanciação de frameworks.
- Alternativas: alternativas correspondem a cada valor dos fatores. No exemplo de instanciação de frameworks, RDL pode ser considerada um fator.
- Interações: um ou mais fatores podem ter interação entre si. Por exemplo, um fator depende do resultado de outro fator. As interações precisam ser levadas em consideração quando se está planejando o experimento.
- Variações Indesejadas (*Blocking Variables*): são variações não previstas que podem acontecer ao longo de um experimento e que afetam os resultados do mesmo. Como exemplo, sujeitos experimentais podem variar mas originalmente não estava prevista essa variação. Essa variação afetaria os resultados, pois sujeitos experimentais tem diferentes graus de conhecimento e maturidade.
- Experimento elementar (*Elementary Experiment* ou *Unitary Experiment*): considerando o exemplo de técnicas de instanciação de frameworks, a aplicação de cada uma das técnicas sobre a unidade experimental é um experimento elementar, também chamado de experimento unitário.
- Replicação Externa (*External Replication*): é a replicação de um experimento feita por pesquisadores independentes geralmente com outras configurações e exemplos em relação ao experimento original mas que procuram reproduzi-lo o mais próximo possível do experimento.
- Replicação Interna (*Internal Replication*): é a replicação de um experimento unitário. Se um experimento unitário é replicado três vezes então se diz que o experimento possui três replicações.
- Erro Experimental: a palavra erro tem uma conotação técnica apenas para caracterizar as diferenças de resultados que ocorrem de uma replicação para outra. Estas diferenças são também chamadas de variações experimentais ou ruído.

2.1.2 Variáveis de Resposta em Engenharia de Software

As variáveis de resposta representam as saídas de um experimento, os dados que precisam ser analisados para que se tornem informações significativas que contribuam de alguma forma para melhorar a unidade experimental. Variáveis de resposta podem ser informações quantitativas (como por exemplo o tempo economizado na geração de código com o uso de um gerador) e são freqüentemente usadas como sinônimos para métricas em ES [7].

Um experimento pode fornecer como resultados uma série de variáveis de resposta. A correta análise dessas respostas depende dos objetivos estabelecidos para o experimento e se baseia em métricas para avaliar os objetivos definidos. Por isso Victor Basili definiu a abordagem GQM (Goal/Question/Metric). GQM é o mecanismo utilizado dentro da Fábrica de Experiências de Basili [8], como parte do QIP [8], para definir e avaliar uma série de objetivos operacionais usando métricas. Foi inicialmente utilizado em uma série de projetos da NASA [11] (NASA Goddard Space Flight Center) para avaliar defeitos encontrados nesses projetos sendo esses não necessariamente projetos de software. Com a expansão de GQM para a Fábrica de Experiências, pode-se utilizar também a abordagem GQM para o ambiente de desenvolvimento de software.

GQM envolve definir o objetivo do experimento e gerar um conjunto de questões. As respostas a essas questões se dão na forma de métricas que por sua vez ajudarão a avaliar se o objetivo do experimento foi atingido ou não.

A Figura 6 mostra o relacionamento entre os objetivos (goals), as questões (questions) e as métricas (metrics).

A aplicação de GQM resulta em um sistema de medidas referentes a um conjunto de problemas e uma série de regras utilizadas para a interpretação dos dados coletados. Esse modelo tem três níveis [11]:

- **Nível Conceitual:** Um objetivo (o *goal* de GQM) é definido para um objeto de estudo, relativo a um ambiente em particular e sob vários pontos de vista. Exemplos de objetos são Produtos (Ex.: Artefatos de software, documentos produzidos durante ciclo de vida, especificações, casos de teste, etc), Processos (Ex.: coleta de requisitos, criação de especificações e fases de testes) ou Recursos (Ex.: Ambientes de escritório e hardware).
- **Nível Operacional:** Um conjunto de questões (*questions* de GQM) é utilizado para caracterizar como um determinado objetivo pretende ser atingido ou não, dependendo do ponto de vista selecionado.
- **Nível Quantitativo:** Um conjunto de dados coletados (*metrics* do GQM) está associado com cada questão com o objetivo de responder de uma forma quantitativa.

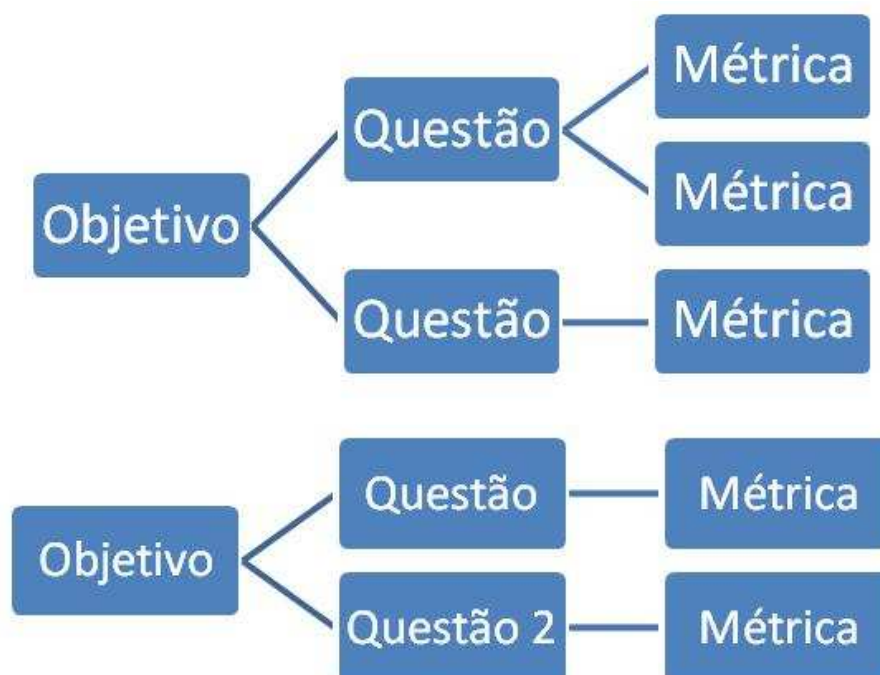


Figura 6 – Relacionamento entre Objetivos, Questões e Métricas em GQM

[11]

A abordagem GQM se tornou um instrumento de grande utilidade para a definição do objetivo da presente pesquisa bem como as questões que se buscou responder e as formas de medidas associadas a cada questão.

2.1.3 Experimentação em Ambientes Reais

O principal motivo para se executar experimentos é tornar conhecidas informações que antes estavam escondidas e transformar dados não compilados em informações úteis e que auxiliam os profissionais e pesquisadores de ES a entender mais e melhor o ambiente ao qual estão inseridos. Entretanto, para que essas informações sejam o mais próximas possível da realidade faz-se necessário executar experimentos em ambientes aproximados da realidade. Experimentos controlados [12] são geralmente executados em ambientes onde existe pouca variação, muitas vezes com estudantes resolvendo pequenas tarefas pré-definidas e anotando informações em formulários. Nesses tipos de experimento não há realismo, ou seja, não são tarefas desempenhadas por trabalhadores reais, enfrentando problemas reais e tampouco o ambiente onde se está realizando o experimento controlado é real. Fatores como tempo para realização de tarefas, distrações no ambiente, pressão e prazos não estão presentes.

Basili, entretanto, pergunta: "Onde estão os laboratórios para Engenharia de Software?" [8]. Eles estão onde aplicações estão sendo desenvolvidas, ou seja, na indústria. Mas não basta ape-

nas que um experimento seja executado em um ambiente real se não forem levadas em consideração conceitos como parâmetros, sujeitos experimentais, variações provocadas, etc, ou seja, muitos fatores podem variar em um ambiente real enquanto que em um experimento controlado esses fatores sofrem pouca ou nenhuma variação. Mais ainda, em ambientes reais existem desafios [12] com relação às tarefas realistas, os sujeitos experimentais realistas e o ambiente realista:

Tarefas Realistas: muitas das tarefas em experimentos com frequência não espelham as tarefas de um ambiente real no que diz respeito ao tamanho da tarefa, complexidade e duração das mesmas.

Sujeitos Experimentais Realistas: a seleção dos sujeitos que farão parte do experimento é um outro desafio. Quando comparados a profissionais da área de ES, estudantes possuem diferentes níveis de conhecimento e podem usar ferramentas diferentes das utilizadas nas empresas.

Ambiente Realista: mesmo quando se consegue tarefas realistas e sujeitos realistas, o ambiente pode não retratar o dia-a-dia de uma corporação. Entretanto, aspectos como os processos utilizados pelos profissionais, as metodologias e as ferramentas precisam ser considerados para se manter o ambiente mais próximo da realidade.

Um estudo [12] apresenta algumas alternativas para trazer o realismo para a execução do experimento. Com relação à obtenção de sujeitos experimentais que sejam profissionais é sugerido:

- Oferecer aos profissionais de uma organização parceria (co-autoria) na escrita do artigo que irá revelar os resultados do experimento para incentivar a participação.
- Oferecer à organização cursos internos e usar esses cursos como experimentos, coletando dados da execução dos exercícios do curso.
- Pagar à organização pelo tempo em que os profissionais se envolveram no experimento.
- Incentivar estudantes a trabalhar na organização mesmo que não seja em turno integral pois os mesmos estariam usando ferramentas do dia-a-dia da organização e seguindo processos reais.

Dessa forma, em vez de puramente estudantes estarão envolvidos profissionais que utilizam ferramentas e processos similares aos da organização, trazendo o experimento mais próximo da realidade.

Com relação ao ambiente realista as considerações são as seguintes:

- Aconselhar os profissionais que irão atuar como sujeitos experimentais não falarem uns com os outros durante o experimento.

- Garantir aos sujeitos que a performance individual será mantida em sigilo. Os sujeitos têm de procurar agir o mais natural possível como se estivessem trabalhando e o fato de suspeitarem que suas performances pudessem estar sendo avaliadas os motivaria a agir diferente quando da execução do experimento.
- Monitorar o experimento, estando disponível para responder questões.
- Garantir à organização e aos sujeitos que serão informados dos resultados do experimento e fazê-lo no tempo apropriado.
- Fornecer aos sujeitos um pequeno treinamento para garantir que os recursos computacionais estejam funcionando apropriadamente.

Note que a correta assistência à organização que aceitou executar o experimento e também o auxílio aos sujeitos são fatores importantes durante a execução do experimento. Com relação à escolha de tarefas realistas, pode-se usar uma maneira sistemática de definir as tarefas através de uma análise dos tipos e frequências das tarefas pois tarefas em experimentos controlados podem durar algumas poucas horas enquanto em ambientes reais algumas tarefas podem durar dias. É preciso, então, fazer um levantamento das tarefas para se criar uma base de conhecimento das mesmas antes de escolher quais serão utilizadas dentro do experimento.

2.2 Porque não são Realizados Experimentos em Engenharia de Software?

É importante para a área de ES entender alguns dos motivos [13] pelos quais se faz pouco uso da experimentação apesar dos enormes ganhos que a realização de experimentos pode trazer. As razões são as mais diversas, variando desde treinamento e entendimento da importância da realização de experimentos até fatores como custo. A seguir são discutidos alguns dos motivos pelos quais existe um ceticismo em relação à experimentação em ES [13].

- Compreensão da importância da experimentação: na área de desenvolvimento de software parece existir um sentimento de que experimentação se aplica muito bem mas somente nas ciências como a física e a medicina. A falta de uma compreensão em relação à experimentação em ES tem origem na falta de treinamento e ensino dos métodos científicos. Como os desenvolvedores de software não tem familiaridade com tais técnicas, o uso da experimentação se torna algo distante, não compreendido como a validação de teorias poderia trazer fatos para o dia-a-dia e auxiliar na tomada de decisão.
- Falta de conhecimento para analisar os dados: além da falta de treinamento e educação em prol da experimentação há também a falta de entendimento estatístico. Uma vez que os

dados foram coletados, como organizá-los para produzir fatos relevantes que contribuam para uma validação?

- Falta de literatura para ES: como a experimentação em ES não é uma prática largamente adotada, o reflexo está na falta de literatura apropriada, principalmente de livros que tragam exemplos de validação na área. Pesquisadores poderiam usar esses livros como instrução desde que eles contivessem exemplos dentro de ES, pois é difícil extrair um conceito olhando um resultados de, por exemplo, validação de alguma droga na medicina.
- Complexidade do ambiente de desenvolvimento de software: uma outra razão para o fraco uso de experimentação é o número de fatores e variáveis relacionados ao ambiente de desenvolvimento de software. Embora isso seja verdade, não fazer uso de experimentos (mesmo em ambientes complexos) pode aumentar os problemas uma vez que deixamos de colher informações relevantes para entender melhor o ambiente ao qual estamos inseridos.
- Dificuldades na obtenção de verdades globais: é difícil na ES obter respostas do tipo "a técnica A é melhor que a técnica B". Entretanto, respostas do tipo "A é melhor que B nas situações X, Y e Z" é melhor do que a falta de conhecimento sobre as técnicas e diminui a incerteza.
- Fatores Humanos no desenvolvimento de software: executar um experimento e repeti-lo em ES pode trazer diferentes resultados devido a aspectos relacionados às pessoas como a motivação. ES não é uma disciplina em que os resultados são adquiridos sem levar em consideração as pessoas. Embora esses fatores existam, não podem ser considerados uma barreira para a realização de experimentos. Entretanto, esses fatores precisam ser minimizados ao máximo para que não ocorram distorções nos resultados.

A Tabela 1 mostra algumas falácias e refutações [7] em relação à experimentação em ES.

A falta de uma cultura de realização de experimentos em ES vem também da falta de consciência da importância dos resultados que a validação de uma ou mais técnicas, processos, ferramentas, etc poderiam trazer ao dia-a-dia dos profissionais e pesquisadores da área. Aceitamos novas verdades simplesmente porque alguém disse que pode funcionar ou, ainda pior, que pode funcionar para todas as situações. Será que aceitaríamos sermos medicados por uma nova droga simplesmente por que alguém disse que essa droga "deve" funcionar? Essa falta de cultura precisa ser modificada e uma das formas de modificar essa cultura é justamente realizando experimentos que tragam fatos ao cotidianos de ES, contestando e validando supostas verdades.

Tabela 1 – Falácias e refutações sobre experimentação em ES

Falácia	Refutação
Métodos científicos tradicionais não são aplicáveis	Para um melhor entendimento, Cientistas da Computação devem observar fenômenos, testar e formular explicações. Tudo com base em fatos
O nível atual de experimentação na área está suficiente	Dados mostram que apenas um percentual muito baixo de cientistas da computação validam o que pregam
Experimentos são caros	Experimentos baratos podem entrar no orçamento de projetos. Experimentos mais caros se pagam através dos resultados que trazem
Demonstrações são suficientes	Demonstrações podem incentivar questionamentos mas meramente ilustram um potencial, não uma verdade
Experimentação torna o progresso mais lento	Com a aplicação cada vez mais freqüente de experimentos poderemos ter justamente o contrário

2.3 Problemas e Considerações quanto a Replicação de Experimentos em Engenharia de Software

Existem alguns fatores que precisam ser levados em consideração quando da aplicação de experimentos em ES. Tais fatores se tornam críticos, principalmente, quando da replicação do mesmo experimento. Aplicativos de software não são produtos que seguem um processo puramente industrial como produtos gerados pela manufatura. Várias aplicações de software com a mesma finalidade podem ter sido produzidas de diferentes maneiras, utilizando processos diferentes de construção e variando até mesmo o ambiente ao qual o software foi construído: Aspectos físicos como mobiliário, condições de temperatura, motivação de recursos humanos, etc. Estes aspectos precisam ser levados em consideração na execução do experimento e também na replicação do mesmo [6].

Um estudo [6] analisou o quanto a falta de **comunicação** entre os diferentes executores de replicação de um experimento prejudica a própria replicação do mesmo. A replicação de um experimento é considerada um sucesso [6] quando as diferentes replicações conseguem achar resultados que agregam valor ao experimento original, sejam eles convergentes ou divergentes. Basicamente, existem dois modelos de compartilhamento das informações [6] nas diferentes replicações de experimentos:

- *Replication Package (RP)*: é a documentação necessária para executar a replicação.

- *Communication Model (CM)*: comunicação entre os grupos do experimento original e os grupos que replicaram o experimento.

Os problemas descritos a seguir foram encontrados na replicação de um experimento que analisava a eficiência das diferentes técnicas de avaliação/inspeção de código. O trabalho replicou um mesmo experimento três vezes, variando o nível de comunicação existente entre os participantes desses três experimentos e analisando os problemas encontrados de acordo com cada nível de comunicação. Os participantes dos experimentos estavam em diferentes localizações e cada grupo replicou o experimento tendo em mãos mais ou menos detalhes a respeito do experimento original. No experimento original, um código com falhas foi provido para que se pudesse executar as diferentes técnicas e medir sua eficiência. A seguir são analisados os problemas e propostas de soluções de acordo com cada tipo de CM [6] na replicação. Vale ressaltar que o trabalho realizado não julgou importante mostrar quais os resultados de cada replicação e sim o quanto o tipo de comunicação afetou a replicação e interferiu nos resultados.

2.3.1 Replicação com Comunicação Zero

O grupo que replicou o experimento recebeu o RP original mas não teve qualquer tipo de comunicação com o grupo que inicialmente executou o experimento. O RP consistiu de:

- Formulários de coleta dos dados.
- Especificação do experimento e programas de treinamento.
- Código fonte com falhas para o propósito do experimento e dos programas de treinamento.
- Planilhas de instruções de avaliação por programa e por técnica de avaliação.
- Descrição das falhas no experimento e nos programas de treinamento e os problemas que essas falhas causavam
- Soluções para cada programa de treinamento, com técnicas diferentes para cada um.

Com o RP original em mãos e sem efetuar comunicação alguma com o grupo que efetuou o experimento, o grupo da replicação partiu para efetuar a replicação do experimento original. A Tabela 2 mostra alguns dos problemas encontrados provenientes da falta de comunicação e mostra sugestão de solução para cada problema encontrado.

Uma das principais lições aprendidas ao replicar um experimento sem comunicação entre a equipe original e a equipe que está replicando o experimento é justamente que é extremamente vital estabelecer um canal de comunicação entre as equipes mas é importante também não ter uma comunicação desordenada ou originada sob demanda.

Tabela 2 – Problemas e soluções para Replicação com Comunicação Zero

Fase	Problema	Proposta de solução
Definição/Planejamento	Definição e planejamento não presentes no RP	Incluir no RP
Execução	Execução das tarefas não definidas no RP	Incluir no RP
Execução	O código sem falhas não foi provido no RP	Incluir código sem falhas
Análise dos dados	Análises originais não faziam parte do RP	Incluir no RP

2.3.2 Replicação com Comunicação Ágil

O próximo CM apresenta resultados ao se agregar mais um canal de comunicação ao processo de replicação utilizando comunicação pontual e organizada. Um novo grupo replicou o experimento original só que dessa vez aplicando um novo CM [6], a Replicação com Comunicação Ágil. Esse CM consistiu de:

- Reunião de *Kick-off*: A reunião de *kick-off* pode ser definida como uma reunião inicial, um "pontapé" inicial. Esse tipo de reunião foi realizada na fase inicial de definição e planejamento da replicação, servindo, então, para analisar o contexto da replicação.
- Surgimento de dúvidas: foi estabelecido que eventuais dúvidas surgidas durante a replicação iriam ser questionadas por telefone e por e-mail. Muitas delas, nessa segunda replicação, foram feitas por e-mail.
- Reunião de agregação: feita após a coleta dos dados e análise de resultados, a reunião serviu para comparar os dados de ambas as replicações e analisar as convergências e divergências.

Com relação ao RP, o mesmo evolui com base nos problemas detectados na primeira replicação, tendo como fonte de melhoria os problemas e soluções encontrados na Tabela 2. Dentre as várias melhorias destacaram-se a adição de *scripts* para descrever os passos de execução da replicação, descrição detalhada das falhas ocorridas nos programas e exemplos de como preencher os formulários de coleta de dados.

Ao aplicar a Comunicação Ágil na replicação do experimento não foram detectados problemas nas fases de definição e planejamento nem na agregação dos resultados. Entretanto, conforme descrito a seguir e sumarizado na Tabela 3, alguns problemas foram encontrados na execução do experimento.

- Atmosfera experimental: os responsáveis pelo experimento ficaram com dúvidas em relação ao comportamento que deveria ser tomado quando da execução do experimento no

que diz respeito aos estudantes que aplicaram as técnicas. Isso indica que o *script* provido no RP estava incompleto pois deveria conter esse tipo de informação.

- Dúvidas sobre como preencher os formulários: os estudantes que executaram o experimento tiveram dúvidas de como preencher os dados no formulário de coleta de informações. Apesar de o RP conter um exemplo de formulário, não foi mencionado que os exemplos deveriam ter sido explicados aos estudantes. Como sugestão, instruções foram adicionadas ao script de execução sobre a explicação inicial que deveria ser dada. Outra sugestão era rever o RP na reunião de *kick-off*.
- Dúvidas com relação ao tempo de execução: os sujeitos (estudantes que executaram o experimento) não tiveram tempo para concluir algumas das técnicas. O tempo não havia sido discutido exaustivamente na reunião de *kick-off*.

A Tabela 3 mostra os problemas encontrados na fase de execução.

Tabela 3 – Sumário dos problemas na execução do experimento com Comunicação Ágil

Fase	Problema	Proposta de solução
Execução	Atmosfera experimental	Incluir no RP como interagir com os sujeitos
Execução	Dúvidas de como preencher formulários	Incluir no RP a explicação
Execução	Dúvidas em relação ao tempo	Discutir na reunião de <i>kick-off</i>

A reunião de agregação, além de servir para analisar e comparar os dados e resultados das replicações já efetuadas, serviu também para a discussão dos problemas que foram encontrados na fase de execução. Dentre os tópicos discutidos na reunião de agregação, foram encontrados alguns pontos que devem ser levados em consideração durante a replicação de experimentos. Um dos pontos refere-se a **treinamento**. Um bom treinamento para os sujeitos indicando como se comportar no decorrer do experimento, quais técnicas poderiam ser aplicadas, detalhes do tempo de execução teria minimizado os problemas encontrados. Um outro ponto importante a ser considerado é a **motivação**. Que motivação se dá a sujeitos de forma que o experimento seja executado com a devida seriedade para que os resultados finais não sejam comprometidos? Na primeira replicação, os sujeitos, após aplicarem as técnicas aprendidas, recebiam *feedback* de aprovação ou reprovação. Na segunda replicação, não existia o conceito de aprovação ou reprovação, ou seja, os sujeitos poderiam aplicar as técnicas mas não ficavam sabendo se foram aprovados ou reprovados no experimento simplesmente pela inexistência desse conceito. O resultado foi que na reunião de agregação foram encontradas diferenças entre a performance e aplicação das técnicas por parte dos sujeitos com relação aos resultados da primeira replicação. As lições que ficam da aplicação desse segundo CM incluem: Comunicação através de reuniões de *kick-off* e agregação para explicações de fatores como tempo; treinamento adequado para explicar as diferentes técnicas e motivação adequada dos sujeitos para que os resultados não sejam afetados por uma execução imprópria do experimento.

2.3.3 Exercitando com Comunicação Ágil

O último CM aplicado utilizou de algumas das práticas adotadas na replicação anterior como o uso de telefone ou e-mail para respostas às questões em relação ao experimento original e reunião de agregação ao final. Entretanto, algumas modificações foram feitas para se adaptar ao contexto na nova replicação.

- Modificações nas técnicas aplicadas: devido ao tempo limitado, a técnica de revisão de código foi deixada de lado nesta replicação.
- Computadores com capacidade aquém do esperado: com recursos de hardware aquém da capacidade desejada para a replicação, o escopo foi modificado pois uma das variáveis de resposta foi deixada de fora. No caso, os casos de teste não foram executados.
- Diferenças no nível de experiência dos sujeitos: os estudantes que estavam atuando como sujeitos dessa replicação tinham um nível de experiência com linguagens de programação inferior aos estudantes de outras replicações. Embora isso pudesse afetar os resultados colhidos da aplicação das técnicas de inspeção de código, também serviu para estudar o efeito dos diferentes graus de conhecimento sobre os resultados do experimento.
- Número de sessões de revisão de código: o número de sessões para inspeção de código foi reduzido mas o tempo de cada sessão foi aumentado. Isso permitiu verificar se, durante uma sessão mais longa, o cansaço afetaria os resultados do experimento.

Além dessas modificações, algumas práticas também não foram adotadas como é o caso da reunião de *kick-off*, por julgarem a reunião não ser crítica para o sucesso da replicação. A Tabela 4 sumariza os problemas encontrados nessa terceira replicação. Vale observar que, como descrito anteriormente, algumas modificações foram feitas e variáveis foram alteradas podendo, assim, afetar os resultados.

Tabela 4 – Sumário dos problemas na execução do experimento com Comunicação Ágil

Fase	Problema	Proposta de solução
Definição/ Planejamento	Mais modificações que o necessários foram feitas	Mais comunicação deveria ser utilizada
Execução	Material do treinamento não foi adequado	Mais comunicação sistemática
Execução	Detalhes faltando no script do experimento	Mais comunicação sistemática
Execução	configuração do experimento não descrita	Incluir no RP e mais comunicação sistemática

Com relação à fase de Definição/Planejamento, foi visto que mais comunicação deveria ter sido utilizada. Esse já foi um problema detectado quando o CM era de comunicação zero e

corrigido no CM de comunicação ágil. As mudanças feitas poderiam ter impacto significativo nos resultados da replicação e poderiam ter sido discutidas para evitar tais modificações.

Uma das lições aprendidas durante a terceira replicação do que a reunião de *kick-off* não poderia ter ficado de fora, pois sem ela mudanças significativas foram feitas que afetaram os resultados e que poderiam ter sido discutidas no início.

Conclusões em Relação à Comunicação para Replicação de Experimentos

O estudo concluiu a extrema importância da comunicação para a replicação de experimentos. Das boas práticas retiradas do artigo que apresenta os tipos de comunicação na replicação pode-se destacar:

- Reunião de *kick-off*.
- Reunião de agregação dos resultados.
- Canal aberto por e-mail e telefone entre os replicadores de um experimento e os pesquisadores que aplicaram o experimento original.

Além dessas práticas, destaca-se ainda alguns fatores que precisam ser levados em consideração como é o caso da experiência dos sujeitos (junior, pleno, sênior), dos recursos computacionais e da motivação dos sujeitos. A devida atenção dada a esses fatores aliada às boas práticas documentadas não garantem o sucesso de uma replicação mas com certeza ajudam a melhorar os resultados ou, no mínimo, forçam os praticantes do experimento a debater sobre os passos da execução.

3 RDL - Reuse Description Language

Este capítulo apresenta uma visão geral sobre RDL (*Reuse Description Language*), uma das maneiras de utilizar um processo disciplinado para a instanciação de frameworks e que será objeto de avaliação em um experimento. A RDL [1] é uma linguagem que especifica os passos em um processo de instanciação de frameworks. Entende-se por instanciação de frameworks a tarefa de estender um determinado conjunto de classes e/ou interfaces usando seus pontos de extensão (ou pontos de variabilidade) para atingir uma necessidade específica, ou seja, criar um modelo final (no caso da RDL, no formato XMI [14]) com os artefatos finais de uma aplicação que estende e faz uso do framework em questão. RDL ajuda a resolver os seguintes problemas:

- O tamanho e complexidade de determinado framework pode dificultar o trabalho de um reutilizador em compreender todos os pontos de extensão disponíveis.
- O processo de instanciação pode requerer determinado fluxo na criação desses artefatos como, por exemplo, determinado conjunto de classes sendo criadas antes de outras classes.
- O desenvolvedor que está instanciando determinado framework precisa fornecer informações adicionais ao processo de instanciação como os nomes de artefatos a serem criados como resultado da instanciação (nomes de classes, métodos, etc.).

Para o primeiro problema, um conhecedor do framework pode criar o script RDL contendo todos os pontos de extensão possíveis de determinado framework. No segundo problema, um criador de um framework (ou alguém que o conheça) pode definir esse processo para que um reutilizador (o desenvolvedor que irá estender o framework) siga o fluxo necessário para estendê-lo na ordem requerida. E com RDL ainda é possível que exista interação com o desenvolvedor no processo de instanciação, o que resolve o terceiro problema. Com todas essas características, um ou mais desenvolvedores não precisam conhecer em detalhes o framework a ser estendido pois o script RDL provê todo o suporte à extensão dos pontos de variabilidade. A necessidade do criador ou mantenedor do framework estar presente também se torna irrelevante uma vez que o script RDL foi criado justamente explorando os pontos de extensão do framework.

Através da ferramenta Reuse Tool, explicada a seguir, scripts RDL são utilizados para auxiliar o reutilizador a criar os artefatos necessários à extensão de um framework, resultando em um modelo final (um arquivo no formato XMI) contendo o framework com a adição dos artefatos criados.

3.1 A Ferramenta Reuse Tool

A ferramenta Reuse Tool tem como objetivo auxiliar o desenvolvedor no processo de instanciação de um framework através da execução de scripts RDL. A Figura 7 mostra as atividades do processo de instanciação de um framework com RDL, desde sua concepção até o modelo final gerado e estendido. Três atores fazem parte desse processo, onde:

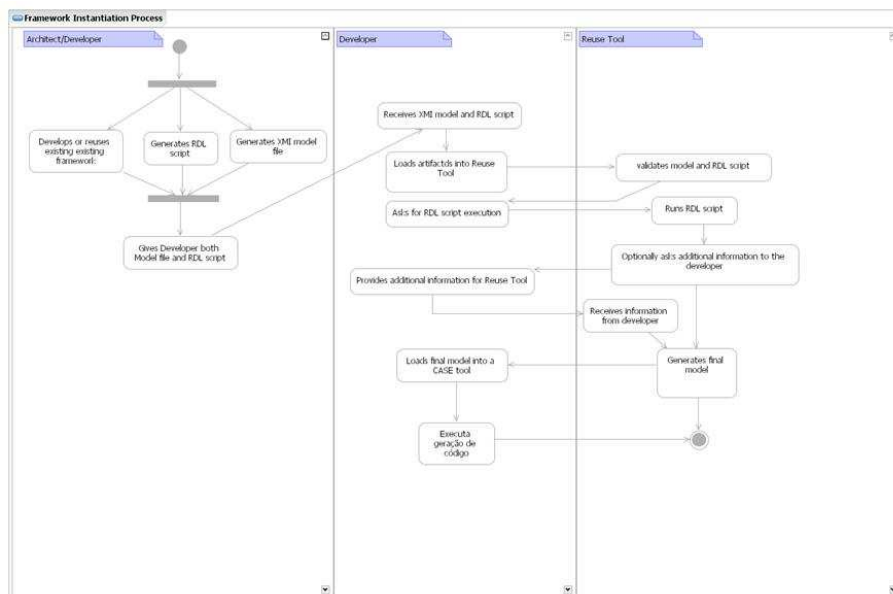


Figura 7 – Atores e atividades do processo de instanciação

- **Arquiteto:** o papel do Arquiteto ou Desenvolvedor é especificar o framework a ser estendido ou utilizar um framework existente. Esse framework deve ser carregado em uma ferramenta UML para posterior exportação no formato XMI [14]. O Arquiteto ou Desenvolvedor ainda cria o script RDL que irá explorar todos os pontos de variabilidade do framework, possivelmente interagindo com o desenvolvedor que irá executar o script. Após a criação desses artefatos, tanto o script RDL como o modelo em formato XMI [14] são enviados ao desenvolvedor que irá executar o script RDL na ferramenta Reuse Tool.
- **Desenvolvedor:** o Desenvolvedor recebe um modelo XMI e um script RDL. Esses artefatos são carregados na ferramenta Reuse Tool para que o processo de instanciação seja iniciado. O Desenvolvedor ainda interage com a ferramenta, provendo informações como nomes de classes (novas ou que estendam as já existentes), nomes de métodos (novos ou que sobrescrevam/sobrecarreguem algum método existente), etc., ou seja, o desenvolvedor fornece todas as informações necessárias para que a ferramenta Reuse Tool gere um modelo final no formato XMI já com os pontos de extensão do framework devidamente estendidos.
- **Reuse Tool:** a ferramenta Reuse Tool (Figura 8) recebe como entrada um modelo no

formato XMI contendo o framework a ser estendido e um script RDL com todos os passos necessários à extensão do framework. A Reuse Tool valida o formato do modelo XMI e a sintaxe do script RDL. Após a validação, o desenvolvedor pode invocar na ferramenta a geração do modelo final (Figura 9).

Cabe observar que o produto final não é código gerado e sim um modelo contendo a aplicação final (Figura 9) já utilizando todos os pontos de extensão do framework. Esse modelo pode facilmente ser inserido dentro de uma ferramenta com suporte a importação de um modelo XMI para posterior geração de código na linguagem orientada a objetos que o desenvolvedor preferir.

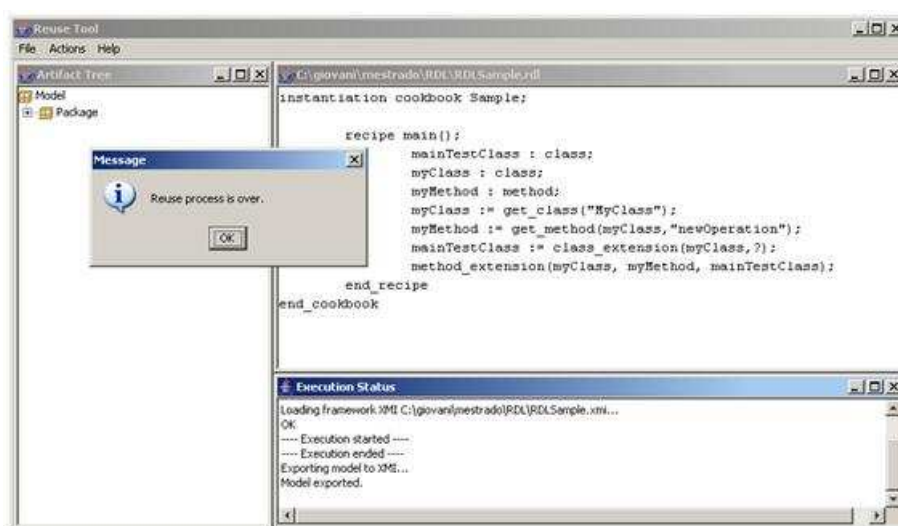


Figura 8 – A ferramenta Reuse Tool executando scripts RDL sobre um modelo XMI

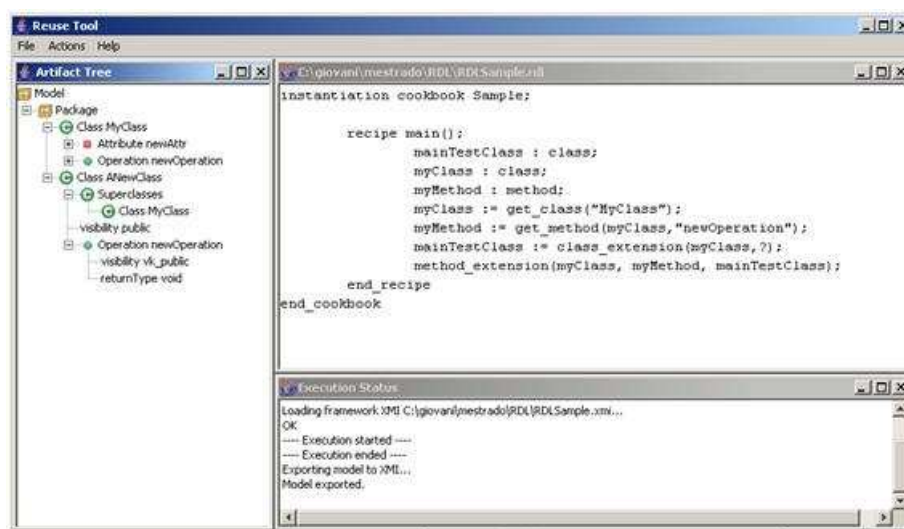


Figura 9 – Modelo final gerado e estendido

3.2 Limitações em RDL

Ao se fazer o primeiro conjunto de testes com a atual versão [15] da RDL foram constatadas algumas limitações importantes para se prosseguir com a idéia de um experimento envolvendo RDL. Essas limitações encontradas se tornaram fator bloqueante da condução de um experimento com a linguagem no estado em que se encontra, pois todas as limitações interferiram diretamente no uso da RDL, uma vez que dizem respeito a componentes essenciais nos frameworks para desenvolvimento de aplicações.

3.2.1 Falta de Suporte a Interfaces

O uso de interfaces em frameworks de aplicações de software ajuda a esconder detalhes de implementações ao prover uma camada de abstração, permitindo, assim, o desacoplamento entre os clientes de determinada funcionalidade e o provedor dessa funcionalidade. Ainda, interfaces geralmente especificam os pontos de variabilidade de determinados frameworks, ou seja, pontos onde pode ocorrer por parte dos desenvolvedores alguma extensão para prover nova funcionalidade ou estender funcionalidade existente. Um exemplo disso são frameworks que utilizam abordagens como injeção de dependência [16]. Tais frameworks dependem das interfaces para prover o desacoplamento total entre o cliente e a implementação dessas interfaces. No código cliente, são declarados objetos como sendo do tipo da interface, mas são injetados (daí o nome Injeção de Dependência) por esses frameworks as implementações dessas interfaces. Outro uso de interfaces diz respeito ao estabelecimento de um contrato de uso, pois linguagens como Java forçam a implementação das operações de um interface para as classes que as implementam.

Ao tentar se modelar um framework de exemplo foi constatado que RDL não suportava interfaces, ou seja, mesmo que o modelo XMI contivesse uma ou mais interfaces, a linguagem não possuía comandos específicos para se obter referência a essas interfaces e assim criar artefatos que implementassem as interfaces. Essa limitação foi considerada de alta criticidade.

A Figura 10 ilustra um diagrama de classes contendo interface e esse mesmo diagrama após o processo de instanciação com RDL na versão sem suporte a interfaces.

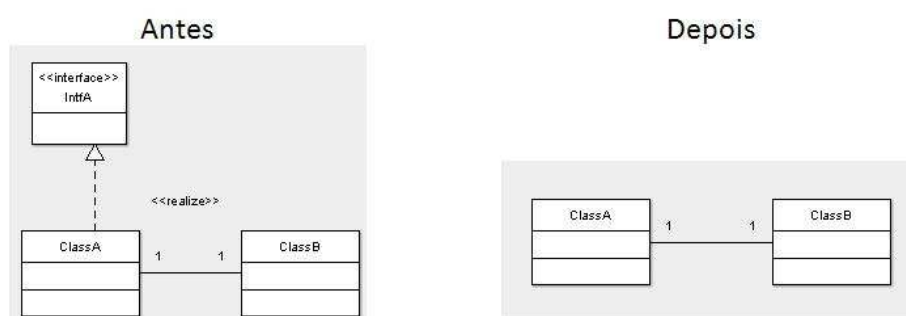


Figura 10 – Problema com interfaces na RDL

Conforme a ilustração da Figura 10, RDL simplesmente não considerava interfaces como parte do processo de instanciação.

3.2.2 Falta de Suporte a Parâmetros em Operações

Um problema básico encontrado na RDL foi a falta de suporte a parâmetros de qualquer natureza em operações das classes de um modelo. Para ilustrar o problema imagine uma classe de nome *A*Class com uma operação nomeada *aMethod* que receba por parâmetro uma informação do tipo *String*. Após executar um script RDL que crie novos artefatos, o novo modelo deve conter a classe *A*Class com sua operação *aMethod* recebendo por parâmetro uma *String*. Entretanto, os parâmetros desapareciam após o processo de instanciação fazendo com que o modelo final ficasse incompleto e extremamente limitado. Ao se observar o código da RDL percebeu-se que realmente toda a parte de carga dos parâmetros de operações não havia sido codificada na sua última versão [15] por conveniência.

Das limitações encontradas essa certamente se tornou a mais grave e a mais limitante para a condução de um experimento real pois a maioria dos frameworks encontrados em ambientes reais de desenvolvimento de software fazem uso constante de parâmetros em operações. A Figura 11 mostra uma classe com operações que contém parâmetros e como resultava essa mesma classe após o processo de instanciação com RDL.

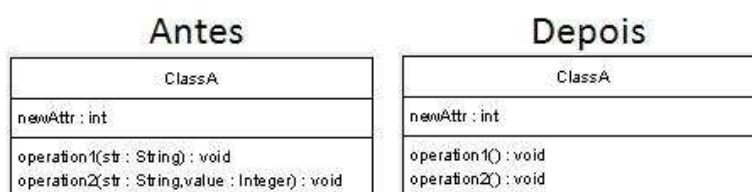


Figura 11 – Problema com parâmetros de operações na RDL

3.2.3 Conclusão das Limitações

As duas limitações relatadas precisavam ser superadas para que o estudo experimental com RDL fosse levado adiante. A seguir, é detalhado o que foi modificado na linguagem para que se eliminasse as duas limitações.

3.3 Melhorando RDL

3.3.1 Suporte a Interfaces

As seguintes modificações foram feitas na linguagem RDL para suportar interfaces:

- Adição de um novo tipo de dados, o tipo *interface*.
- Adição do comando *get_interface*, que obtém uma referência a uma interface dentro de um modelo representado em formato XMI.
- Adição do comando *interface_implementation*, que recebe uma interface por parâmetro e o nome de uma classe que irá implementar essa interface. Retorna um tipo *class* como referência a nova classe que implementa essa interface.

Para que os comandos mencionados pudessem ser acrescentados à linguagem foi necessário modificar o arquivo de especificação da linguagem (Apêndice E – arquivo de especificação da RDL) e recompilar a mesma para produzir novo conjunto de classes e interfaces que suportassem os comandos adicionados. Além disso, precisou-se fazer modificações consideráveis dentro da máquina virtual da linguagem para que os novos comandos pudessem ser devidamente processados. A Figura 12 mostra um script RDL suportando os novos comandos.

```
instantiation cookbook Simple;
  recipe main();

  myintf: interface;
  simple : class;
  newSimple : class;
  classImpl : class;

  simple := get_class("Simple");
  newSimple := class_extension(simple, "SimpleExt");
  myintf := get_interface("MyInt");
  classImpl := interface_implementation(myintf, "ClassImpl");
  new_method(classImpl, "ola");
  new_method(classImpl, "ola2");

end_recipe;
end_cookbook;
```

Figura 12 – RDL com suporte a interfaces

Nota-se a invocação ao comando *get_interface* para obter uma referência a uma interface dentro do modelo de nome *MyInt*. A referência a essa interface é atribuída a uma variável do

tipo interface de nome *myintf*. Após, o comando *interface_implementation* recebe a referência e o nome da classe que irá implementar essa interface, no caso, *ClassImpl*. A variável *classimpl* do tipo *class* recebe a referência para a nova classe que irá implementar a interface *MyInt* no modelo final gerado.

Após essa modificação ter sido feita inúmeros testes foram efetuados em modelos que continham as interfaces. O resultado final foi positivo pois agora RDL exporta a interface e também as classes que implementam essa interface.

3.3.2 Suporte a Parâmetros em Operações

Nessa modificação não foi preciso mexer no arquivo de especificação da linguagem, como aconteceu no caso da adição de comandos para suportar interfaces. Apenas a máquina virtual da linguagem RDL precisou ser modificada para que entendesse parâmetros de operações quando da carga de um modelo XMI e também fosse apta a exportar as mesmas operações com seus devidos parâmetros para o modelo final no formato XMI. Entretanto, apesar de não precisar modificar o arquivo de especificação da linguagem, as mexidas foram volumosas e nova lógica precisou ser adicionada.

Então, a partir das modificações realizadas, buscou-se então modelar o framework e elaborar uma primeira versão do planejamento do experimento. O planejamento do experimento é detalhado no capítulo 4 e o framework utilizado é descrito na seção 4.3.

4 O Estudo Experimental

Experimentos procuram validar teorias existentes e em Engenharia de Software isso não é diferente [7]. Na Engenharia de Software objetivamos usar experimentos para melhorar, aprimorar ou simplesmente obter informações a respeito da validade de ferramentas, processos, modelos, etc. Procuram-se respostas a perguntas tais como "é mais produtivo usar o processo A ou B? Qual ferramenta acelera mais o desenvolvimento de software, a ferramenta A ou B?". No contexto de ferramentas de instanciação de frameworks os aspectos importantes a serem questionados e respondidos dizem respeito à produtividade, número de erros provenientes do processo de instanciação e entendimento do framework em questão. Com a idéia de se obter dados que demonstrem se uma ferramenta de instanciação de framework realmente auxilia o processo de instanciação é que se decidiu fazer um experimento com RDL. Após todas as modificações necessárias na linguagem RDL (conforme detalhado na seção 3.2) partiu-se para o planejamento do experimento. A abordagem utilizada para toda a definição do estudo experimental foi a abordagem GQM [8], já detalhada na seção 2.1. Com GQM estabeleceram-se os objetivos do experimento, as questões relacionadas à pesquisa bem como as métricas.

4.1 Objetivo e Hipóteses do Experimento

4.1.1 Objetivo do Estudo Experimental

Comparar, no processo de instanciação de frameworks, o uso da RDL [1] e técnica manual *com o propósito* de caracterizar o tempo despendido no processo de instanciação, a quantidade de erros encontrados e a quantidade de acessos a documentação de um framework nesse processo por cada uma das abordagens, *com foco* na produtividade e na qualidade, *sob o ponto de vista* do engenheiro de software, *no contexto* da instanciação e desenvolvimento de um projeto de software (*toy example*) a ser demonstrado adiante nesse planejamento de experimento.

4.1.2 Objetivo da Medição

Para definição dos objetivos de medição, foram elaboradas as seguintes questões:

1. O tempo gasto na instanciação do framework utilizando RDL é igual ao tempo gasto

utilizando técnica manual?

2. O número de erros e correções realizadas durante os processos de instanciação e codificação é igual quando se usa RDL comparando com técnica manual?
3. A quantidade de acessos a documentação do framework com RDL é igual à quantidade de acessos a documentação usando técnica manual?

Métricas

Para cada questão de pesquisa definida é importante estabelecer que métricas serão utilizadas para a coleta de dados.

- A métrica associada à Questão 1 corresponde ao esforço medido pela relação do tempo gasto em minutos por cada participante durante a instanciação do framework.
- A métrica relacionada à Questão 2 corresponde ao número de erros e modificações no código durante o processo de instanciação do framework e de codificação do produto final.
- A métrica relacionada à Questão 3 corresponde a quantidade de acessos a documentação do framework durante o processo de instanciação e a codificação do produto final.

A Tabela 5 mostra a relação entre as questões, métricas e a forma de coleta consideradas para o experimento.

Tabela 5 – Relação entre questão, métrica e forma de coleta.

Questão	Métrica	Forma de Coleta
1	Tempo despendido no processo de instanciação.	O pesquisador irá cronometrar o tempo total despendido no processo de instanciação, codificação e correções.
2	Número de erros relacionados ao entendimento da regra de negócio, uso correto do framework ou erros de codificação.	O pesquisador utilizará classes de teste automatizado que serão responsáveis por enviar dados corretos e incorretos para posterior comparação com os resultados esperados. Essas classes deverão ser executadas ao final da codificação por parte do participante. Os resultados serão gravados de forma incremental em arquivos externos à aplicação que servirão de base para análise dos dados referentes à métrica.
3	Quantidade de acessos à documentação do framework.	O pesquisador irá anotar toda vez que os participantes fizerem uso da documentação do framework, interrompendo a atividade de instanciação, codificação ou correção.

Hipóteses do Estudo Experimental

A Hipótese nula é uma afirmação cujo objetivo do experimento é negar [8]. As Hipóteses Alternativas, por sua vez, têm como objetivo contrariar a Hipótese Nula. As hipóteses foram definidas como segue:

Variável Tempo :

Hipótese Nula, H0: O esforço envolvido na instanciação do framework e codificação do produto final utilizando RDL é igual ao esforço utilizando técnica manual.

Medidas: O esforço é medido pela relação do tempo gasto em minutos por cada participante durante a instanciação do framework, ou seja, a diferença entre o tempo final e o tempo inicial de cada abordagem, onde:

Δt_{rdl} : Representa a variação de tempo gasto em minutos para instanciar e codificar utilizando RDL.

Δt_{man} : Representa a variação de tempo gasto em minutos para instanciar e codificar utilizando técnica manual.

$$H0 : \Delta t_{rdl} = \Delta t_{man}$$

Hipótese Alternativa, H1: O esforço envolvido na instanciação do framework e codificação do produto final utilizando RDL é menor do que o esforço utilizando técnica manual.

$$H1 : \Delta t_{rdl} < \Delta t_{man}$$

Hipótese Alternativa, H2: O esforço envolvido na instanciação do framework e codificação do produto final utilizando RDL é maior do que o esforço utilizando técnica manual.

$$H2 : \Delta t_{rdl} > \Delta t_{man}$$

Variável Quantidade de Erros :

Hipótese Nula, H0: O número de erros encontrados durante o processo de instanciação e codificação com RDL é igual ao número de erros com o uso de técnica manual, considerando que um erro pode ser de codificação, sub-utilização ou mau uso do framework ou não atendimento de uma ou mais regras de negócio.

Medidas: O número de erros encontrados no código durante o processo de instanciação do framework e de codificação do produto final.

Erdl: Representa o número de erros encontrados durante o processo de instanciação e codificação com RDL.

Eman: número de erros encontrados durante o processo de instanciação e codificação com técnica manual.

$$H0 : E_{rdl} = E_{man}$$

Hipótese Alternativa, H1: O número de erros encontrados durante os processos de instanciação e codificação usando RDL é menor que o número de erros com o uso de técnica manual.

$$H1 : E_{rdl} < E_{man}$$

Hipótese Alternativa, H2: O número de erros encontrados durante os processos de instanciação e codificação usando RDL é maior que o número de erros com o uso de técnica manual.

$$H2 : E_{rdl} > E_{man}$$

Variável Quantidade de Acessos a Documentação :

Hipótese Nula, H0: A quantidade de acessos aos documentos (documentação dos requisitos e do framework) com RDL é igual a quantidade de acessos utilizando o processo manual.

Ardl: Representa a quantidade de acessos a documentação encontrados durante o processo de instanciação e codificação com RDL.

Aman: Representa a quantidade de acessos a documentação encontrados durante o processo de instanciação e codificação com técnica manual.

$$H0 : A_{rdl} = A_{man}$$

Hipótese Alternativa, H1: A quantidade de acessos a documentação com RDL é menor do que a quantidade de acessos utilizando o processo manual.

$$H1 : A_{rdl} < A_{man}$$

Hipótese Alternativa, H2: A quantidade de acessos a documentação com RDL é maior do que a quantidade de acessos utilizando o processo manual.

$$H2 : A_{rdl} > A_{man}$$

Seleção das variáveis

As variáveis independentes e dependentes [6] foram:

Variáveis independentes:

- Técnicas para instanciação de frameworks.
- Experiência dos desenvolvedores (variável de bloqueio).

Variáveis Dependentes:

- Tempo despendido no processo de instanciação e codificação.
- Quantidade de erros encontrados durante o processo de instanciação e codificação.
- Quantidade de acessos feitos a documentação durante o processo de instanciação e codificação.

4.2 Planejamento do Experimento

4.2.1 Caracterização do Contexto

Processo

O experimento utilizou abordagem In-vitro, na qual o conjunto de participantes executou o experimento em um ambiente controlado, ou seja, não no ambiente natural do participante. Houve o devido acompanhamento do pesquisador para a devida coleta dos dados e demais observações.

Realidade

O problema estudado é um problema de sala de aula e corresponde a um framework para criação de aplicações que suportem o recebimento de artigos para congressos científicos. O framework é detalhado na seção 4.3.

Generalidade

O experimento é específico e com validade apenas no escopo do presente estudo.

4.2.2 Experimento Piloto

Antes da execução do experimento se fez necessária a execução de um experimento piloto nos mesmos moldes do que seria executado, com o objetivo de encontrar eventuais problemas no processo de execução do experimento ou nos artefatos do framework utilizado como a sua documentação ou o código executável. Ainda, validou-se o script RDL e algum problema da própria linguagem. O experimento piloto foi executado com um mestrando da área de ciências da computação com conhecimento na linguagem Java [17].

4.2.3 Instrumentação

Artefatos

Os seguintes artefatos foram fornecidos para os participantes efetuarem o experimento.

Participantes com RDL:

- Um arquivo no formato XMI [14] contendo o framework MySubmission a ser instanciado.
- A ferramenta Reuse Tool [15] para o processo de instanciação com RDL.
- Script RDL (Apêndice C – Script RDL usado no experimento) para explorar pontos de extensão do framework.
- A ferramenta modelagem UML ArgoUML [18] para apoiar a geração de código após o framework ter sido estendido com RDL.
- Ambiente de desenvolvimento Java Eclipse [19] em sua versão 3.2 para a etapa de codificação.
- Classe de teste do framework JUnit [20] em sua versão 1.3 preparada para testar o produto final codificado.
- Documentação técnica do framework MySubmission contendo explicações detalhadas de como implementar sistemas de apoio à submissões de artigos para conferências, bem como diagrama com todas as classes do framework.
- Caso de uso com o problema a ser resolvido.
- Questionário de avaliação do experimento conforme Apêndice A –, sob a forma de entrevista.

Participantes Sem RDL:

- Ambiente de desenvolvimento Java Eclipse [19] em sua versão 3.2 para a etapa de codificação.
- Classe de teste do framework JUnit [20] em sua versão 1.3 preparada para testar o produto final codificado.
- Documentação técnica do framework MySubmission contendo explicações detalhadas de como implementar sistemas de apoio à submissões de artigos para conferências bem como diagrama com todas as classes do framework.
- Caso de uso com o problema a ser resolvido.
- Questionário de avaliação do experimento conforme Apêndice A –, sob a forma de entrevista.

4.3 Definição do Framework

Parte fundamental de um experimento envolvendo instanciação de frameworks é o framework que servirá de base para todo o processo de instanciação. Frameworks são peças importantes para promover o reuso de funcionalidades e prover para aplicações uma arquitetura base que permita pontos de extensão para que novas funcionalidades sejam adicionadas às funcionalidades já providas, funcionalidades essas que podem ser tanto reutilizadas como ter suas capacidades estendidas. A decisão sobre qual framework iria ser utilizado no experimento levou em consideração os seguintes aspectos:

- Não seria utilizado um framework conhecido de mercado. Apesar de terem sua capacidade comprovada, a utilização de frameworks de mercado poderia adicionar um complicador ao experimento uma vez que muitos dos sujeitos experimentais poderiam conhecer as funcionalidades do mesmo e explorariam seus pontos de extensão sem precisar recorrer a documentação, afetando fatores como tempo de instanciação e até mesmo evitando a ocorrência de erros durante a execução do experimento.
- O framework deveria ter as mesmas características dos frameworks encontrados no mercado e utilizados para desenvolvimento corporativo. Características essas como uso de interfaces e classes abstratas para permitir extensão, uso de arquivos de configuração para definição de comportamento e o uso de design patterns tais como os de fábrica [2] para instanciação de objetos do próprio framework ou criados pelo desenvolvedor.

- O framework deveria prover características suficientes para que pudessem ser explorados os principais comandos da RDL como os comandos de extensão, de repetição em laço e interação com o desenvolvedor.
- Embora a utilização de frameworks conhecidos de mercado fosse descartada, o framework a ser utilizado no experimento não deveria fugir das características de mercado, ou seja, se aproximar ao máximo da realidade vivida pelos desenvolvedores no processo real de instanciação para criação de aplicações corporativas.

Dados os aspectos mencionados surgiu a idéia de desenvolver um framework que suportasse a criação de aplicações para submissão de artigos para conferências. O framework iria prover funcionalidades básicas para aplicações desse tipo bem como prover pontos de extensão para que os desenvolvedores pudessem criar funcionalidades customizadas. O framework foi modelado e criado utilizando linguagem de programação Java [17] por ser uma linguagem com suporte a orientação a objetos e que também suportaria os aspectos descritos anteriormente de forma natural.

4.3.1 MySubmission Framework

Ao framework foi dado o nome de **MySubmission**. A Figura 13 mostra o diagrama de classes do framework, que foi modelado e desenvolvido pelo próprio pesquisador para suportar a criação de aplicações de controle de submissões de artigos para conferências.

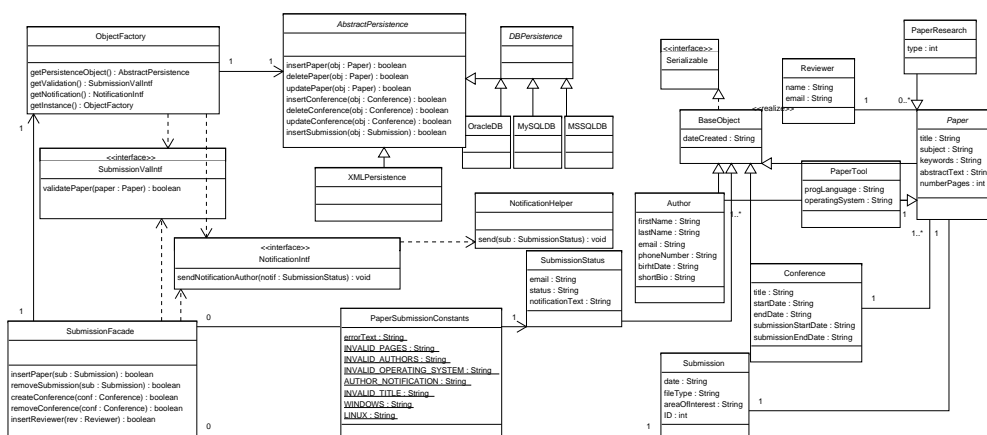


Figura 13 – Diagrama de Classes do Framework MySubmission.

As características especificadas e construídas no framework MySubmission são:

Persistência

O framework desenvolvido para o experimento suporta persistência dos dados para duas fontes de dados: XML e bancos de dados relacionais, sendo que os sistemas gerenciadores de banco de dados suportados são Oracle, MySql e Microsoft SQL Server. A classe abstrata *AbstractPersistence* provê as operações permitidas sobre as fontes de dados suportadas, conforme mostra a Figura 14.

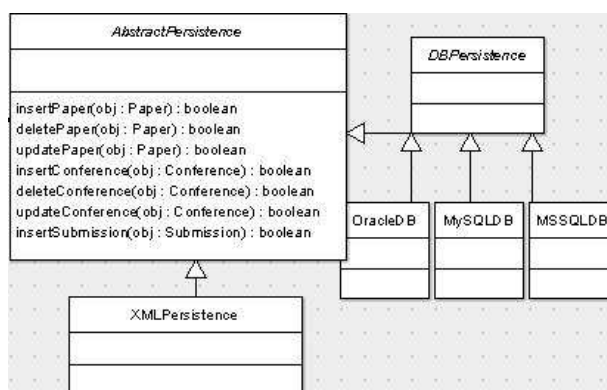


Figura 14 – Classes de persistência.

Fábrica de Instâncias

A classe *ObjectFactory* pode ser classificada como sendo uma implementação do padrão de projetos Abstract Factory [2] pois provê instâncias concretas de classes que implementam a persistência, o envio de notificações e a validação das regras de negócio que precisam ser implementadas na aplicação. Essa classe também é um singleton [2], ou seja, é uma classe que é instanciada somente uma vez quando se deseja ter apenas uma instância da classe em memória. Para que *ObjectFactory* retorne a instância correta das classes é necessário que o desenvolvedor edite o arquivo *PaperSubmission.properties*, colocando o nome da classe concreta que deverá ser retornada. O trecho a seguir mostra um exemplo do arquivo *PaperSubmission.properties*.

```

ConcretePersistenceClass=XMLPersistence
ConcreteValidationClass=ValidationImpl
ConcreteNotificationClass=NotificationImpl
  
```

A Classe de Fachada

A classe *SubmissionFacade* (Figura 15) implementa o padrão de projetos *Façade* [2]. Esse padrão de projetos tem por objetivo esconder de clientes da classe a complexidade envolvida nas operações que estão expostas na classe. É uma porta de entrada para as operações do framework. O desenvolvedor tem que editar essa classe colocando o fluxo de chamadas para validação, notificação e persistência de acordo com as classes criadas e inseridas dentro do arquivo *PaperSubmission.properties*.

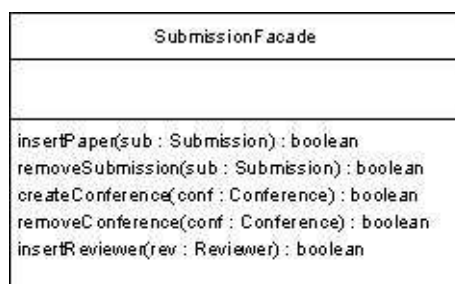


Figura 15 – Classe SubmissionFacade

4.4 Execução do Experimento

Após as extensões efetuadas na linguagem RDL (seção 3.2), do planejamento e piloto efetuados (seção 4.2) partiu-se para a execução do experimento com os participantes. As seções a seguir informam detalhes da execução do experimento.

4.4.1 Participantes

Para a seleção dos participantes do experimento, procurou-se minimizar ao máximo os problemas relatados na seção 2.1, mais especificamente com relação aos ambientes reais para execução de experimentos (seção 2.1.3). Para isso, partiu-se em busca de participantes que sejam trabalhadores da indústria de desenvolvimento de software com experiência prática no desenvolvimento de software corporativo e anos de experiência na linguagem Java. Esses participantes foram selecionados por conveniência, dentro da rede de relacionamentos do pesquisador, observando-se o critério de experiência. A seguir alguns dados demonstrando o perfil dos participantes.

- Total de dezesseis participantes selecionados.
- A média, em anos, de tempo de desenvolvimento de software usando Java para os participantes ficou em seis anos e meio.
- Todos os participantes possuem ensino superior completo.
- Três participantes são mestres em Ciências da Computação.
- Oito participantes possuem alguma certificação Java.
- Todos os participantes são desenvolvedores de software de uma multi-nacional com mais de quatro mil funcionários no setor de Tecnologia da Informação.

Conforme a média de tempo de desenvolvimento de software pode-se dizer que a grande maioria dos participantes é considerada experiente em desenvolvimento de software utilizando linguagem de programação Java.

Tabela 6 – Distribuição dos participantes

Participante	Usou RDL?
1	Não
2	Não
3	Não
4	Não
5	Sim
6	Sim
7	Sim
8	Sim
9	Sim
10	Não
11	Não
12	Não
13	Sim
14	Não
15	Sim
16	Sim

A ordem de execução do experimento seguiu critérios de conveniência, ou seja, nota-se que em seqüência alguns participantes utilizaram a mesma técnica. Isso deu-se ao fato de que a reorganização do ambiente do experimento (preparar o ambiente de desenvolvimento, limpar arquivos de *log*, etc) era facilitada quando o próximo participante iria utilizar a mesma técnica que o anterior. Por isso, optou-se por atribuir a um bloco de participantes a mesma técnica.

4.4.2 Procedimentos de Execução do Experimento

Os seguintes procedimentos eram seguidos quando da execução do experimento com um participante. Os procedimentos variam de acordo com a técnica utilizada com cada um.

Para os que utilizaram RDL os procedimentos foram:

1. Participante chega ao local de execução do experimento.
2. Pesquisador apresenta objetivo do experimento e considerações gerais.
3. Pesquisador efetua treinamento que inclui:
 - Apresentação do caso de uso a ser implementado.
 - Apresentação do framework *MySubmission*.

Apresentação da documentação do framework *MySubmission*.

Apresentação do ambiente de desenvolvimento Eclipse [19] para codificação com o framework *MySubmission* devidamente carregado no ambiente.

Apresentação da ferramenta ArgoUML [18] para manipulação de diagramas UML com framework *MySubmission* já carregado.

Apresentação da ferramenta Reuse Tool [15] para processo de instanciação com RDL e do script RDL com demonstração prática de uso, utilizando um caso envolvendo simples instanciação.

4. Participante inicia execução da instanciação utilizando ferramenta Reuse Tool.
5. Participante carrega na ferramenta ArgoUML o arquivo XMI gerado no passo anterior.
6. Participante gera código Java utilizando ferramenta ArgoUML para as classes que foram criadas como resultado do processo de instanciação com RDL.
7. Participante carrega classes geradas para dentro do ambiente de desenvolvimento Eclipse.
8. Participante inicia codificação do caso de uso no ambiente de desenvolvimento.
9. Participante e pesquisador testam o produto final com a classe de testes já carregada no ambiente de desenvolvimento.
10. Participante termina a execução do experimento.
11. Participante responde questionário de avaliação.

Para os participantes que não efetuaram o experimento com RDL os procedimentos foram:

1. Participante chega ao local de execução do experimento.
2. Pesquisador apresenta objetivo do experimento e considerações gerais.
3. Pesquisador efetua treinamento que inclui:
 - Apresentação do caso de uso a ser implementado.
 - Apresentação do framework *MySubmission*.
 - Apresentação da documentação do framework *MySubmission*.
 - Apresentação do ambiente de desenvolvimento Eclipse para codificação com framework *MySubmission* devidamente carregado no ambiente.
4. Participante inicia codificação do caso de uso no ambiente de desenvolvimento.
5. Participante testa o produto final com a classe de testes já carregada no ambiente de desenvolvimento.

6. Participante termina a execução do experimento.

7. Participante responde questionário de avaliação.

A Tabela 7 mostra os tempos de execução relativos aos procedimentos descritos acima.

Tabela 7 – Tempos de Execução dos Procedimentos

Procedimento	Tempo (min)
Apresentação dos Objetivos	5
Treinamento	5
Execução	60
Questionário Avaliação	10

O tempo de execução mostrado na Tabela 7 diz respeito ao tempo previsto de execução do experimento. Entretanto, a maioria dos participantes terminou antes do tempo previsto.

4.4.3 Coleta dos Dados

As variáveis de resposta (veja seção 2.1.1) precisam ser cuidadosamente coletadas para toda a análise quantitativa e interpretação necessárias. Para cada uma das variáveis relacionadas ao experimento foi adotada uma forma diferente de realizar a sua coleta.

Quantidade de Erros

Conforme definido nas hipóteses (seção 4.1.2) foram considerados erros todo o problema decorrente do não atendimento às regras de negócio especificadas (veja Apêndice B –) ou mau uso do framework, como por exemplo não especificar nos arquivos de *properties* quais classes serão usadas pelo framework.

Partiu-se, então, para uma abordagem automatizada no caso dos erros, ou seja, classes de teste seriam responsáveis por testar o produto final executando uma série de casos de teste para os diferentes cenários com o fim de validar se o requisito solicitado estava sendo atendido e se o framework foi usado corretamente. Uma classe de teste do framework de testes JUnit [20] foi criada e cada operação era responsável por testar um diferente cenário como, por exemplo, passar dados válidos para a classe de fachada a fim de verificar se tudo que foi solicitado estava realmente codificado. Todos os dados decorrentes da execução da classe de teste eram devidamente salvos em arquivos texto para posterior análise e levantamento da quantidade de erros. A listagem a seguir mostra um exemplo de dados gravados em arquivo texto.

```
[main] INFO  MainTest - ::
[main] INFO  MainTest - START - VALID TEST SCENARIO
[main] INFO  MainTest - Expected result = true
```

```
[main] INFO MainTest - Got result = false
[main] INFO MainTest - Any error message? =
[main] INFO MainTest - FINISH - VALID TEST SCENARIO
[main] INFO MainTest - ::
[main] INFO MainTest - START - INVALID TITLE
[main] INFO MainTest - Expected result = false
[main] INFO MainTest - Got result = false
[main] INFO MainTest - Any error message? =
[main] INFO MainTest - FINISH - INVALID TITLE
[main] INFO MainTest - ::
```

Nessa listagem nota-se que dois cenários foram testados. No primeiro, foram passados dados corretos mas houve algum erro de validação. No segundo caso, a validação de presença de título nos artigos estava correta.

Em relação ao uso do framework, como foi solicitado que os dados fossem salvos em formato XML, verificou-se a presença do arquivo salvo para validar se o framework foi usado corretamente. Ainda, o pesquisador fez validação final no código para encontrar algum uso errado das classes do framework.

Quantidade de Acessos a Documentação

Já a quantidade de acessos aos documentos à disposição dos participantes não foi automatizada. O pesquisador observava a execução do experimento e anotava qualquer acesso relevante a documentação (seja acesso ao requisito ou a documentação do próprio framework). Entende-se por acesso relevante toda e qualquer interrupção na execução do experimento a fim do participante validar algo na documentação para seguir seu trabalho. Ainda, vale salientar que os documentos à disposição dos participantes estavam impressos.

Tempo de Instanciação e Codificação

O tempo foi cronometrado pelo pesquisador, separado por tempo de leitura inicial dos documentos, tempo de instanciação (se foi utilizado com RDL) e tempo de codificação das regras de negócio, incluindo aí o tempo de correção de algum erro encontrado pela classe de teste responsável por testar o produto final.

Para efeitos de análise qualitativa, os tempos puderam ser interpretados de forma isolada. Já para a análise quantitativa, considerou-se o tempo total.

4.4.4 Análise dos dados

A análise dos dados aqui apresentada se baseou em tabelas e gráficos apresentados pelo software de análise estatística SPSS [21]. Os testes utilizados para as análises foram os testes de Levene [21] e Shapiro-Wilk [21] para verificação da homocedasticidade e normalidade das amostras, respectivamente. Além disso, se utilizou o gráfico de BoxPlot [21] para verificação de pontos de distorção na amostra (*outliers*). Além da verificação dos outliers, um gráfico BoxPlot mostra também dados referentes ao maior valor observado da população analisada, menor valor observado e a mediana encontrada.

Mas para que se possam fazer as avaliações corretas dos dados obtidos no experimento foi preciso antes fazer algumas verificações para se definir os tipos de análises que seriam empregadas. A primeira verificação realizada foi o teste da normalidade dos dados, ou seja, o quanto os dados coletados seguiam uma curva normal (verificação se as amostras poderiam ser aproximadas por curvas normais e, portanto, passíveis de serem analisadas usando-se um teste paramétrico). O teste de normalidade de Shapiro-Wilk mostrou que os dados, coletados para todas as variáveis, seguem uma curva normal, de acordo com a Tabela 8. Cabe observar que o nível de significância adotado para os dados foi de 5%. Para os testes adotou-se uma hipótese nula e uma alternativa onde:

- Hipótese nula H_0 : A distribuição dos dados é normal.
- Hipótese alternativa H_1 : A distribuição dos dados não é normal.

A Tabela 8 mostra os resultados obtidos para as variáveis **Erro**, **Quantidade de Acessos** e **Tempo**.

Tabela 8 – Teste de Shapiro-Wilk para as Variáveis do Experimento

Variável	Com RDL?	Significância
Erros	Não	0,366
	Sim	0,428
Quantidade de Acessos	Não	0,827
	Sim	0,295
Tempo	Não	0,366
	Sim	0,368

Pelo apresentado na Tabela 8, todos os níveis de significância são superiores a 5% (ou 0,05), indicando que não é possível negar a hipótese nula. Isso indica que os dados podem ser considerados duas amostras da mesma população, ou seja, seguem uma distribuição normal.

Uma outra verificação feita diz respeito a presença de igualdade de variância das amostras (homocedasticidade). Para comprovar essa igualdade utilizou-se o teste de Levene.

Novamente, para os testes adotou-se uma hipótese nula e uma alternativa onde:

- Hipótese nula H0: As variâncias são iguais.
- Hipótese alternativa H1: As variâncias não são iguais.

A Tabela 9 mostra os resultados obtidos para as variáveis Erro, Quantidade de Acessos e Tempo.

Tabela 9 – Teste de Levene para Igualdade das Variâncias

Variável	Igualdade das Variáveis	Significância
Erros	Assumindo	0,256
	Não Assumindo	-
Quantidade de Acessos	Assumindo	0,123
	Não Assumindo	-
Tempo	Assumindo	0,076
	Não Assumindo	-

Considerando os resultados obtidos no Teste de Levene e o grau de significância de 5% como parâmetro, não constatou-se indícios para negar a hipótese nula.

Já a análise de eventuais distorções nas amostras, ou os *outliers*, se deu através dos gráficos de BoxPlot. Na Figura 16 é possível ver a distribuição dos dados e a sua média, indicada pela linha preta no interior do retângulo. O Gráfico diz respeito à variável **Erros**.

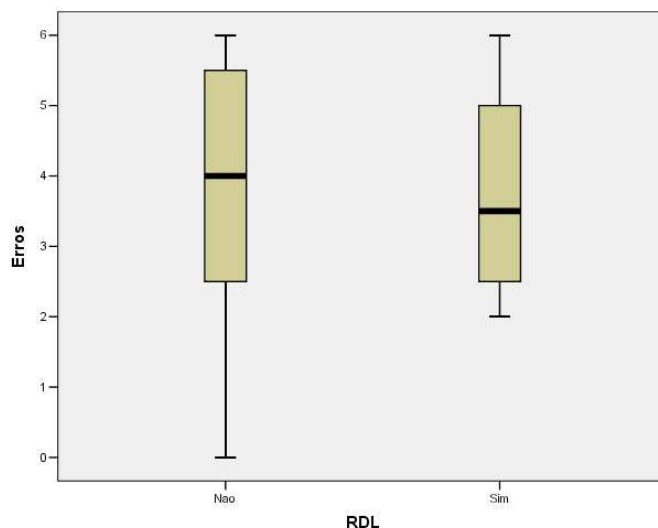


Figura 16 – Gráfico de Boxplot para a Variável Erros

Conforme visto na Figura 16, as médias estão muito próximas uma da outra e não foram identificados *outliers*.

Já a Figura 17 mostra os dados encontrados para a variável **Quantidade de Acessos**.

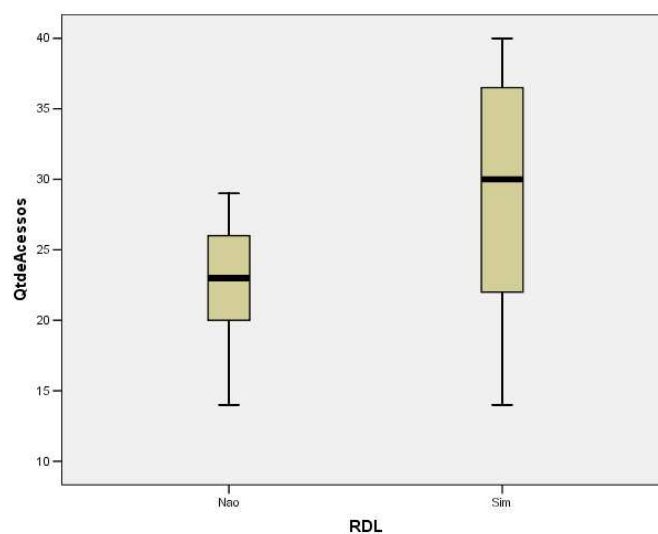


Figura 17 – Gráfico de Boxplot para a Variável Quantidade de Acessos

Nesse caso, observa-se uma dispersão maior para os que usaram RDL para instanciar o framework do experimento mas também não foram encontrados *outliers*.

A última variável, **Tempo** de instanciação/desenvolvimento, pode ser observada na Figura 18.

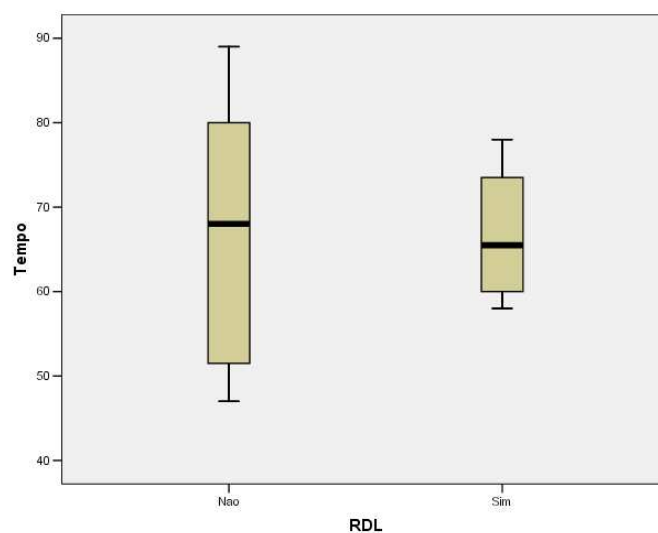


Figura 18 – Gráfico de Boxplot para a Variável Tempo

De forma similar à variável **Erros**, a variável **Tempo** não apresentou diferenças significativas no que diz respeito às médias, representadas pela linha preta dentro de cada retângulo. Também não foram encontrados *outliers* para essa variável. Cabe apenas observar que a variação em relação a tempo foi menor para aqueles que usaram RDL do que aqueles que usaram técnica manual.

Após todas as análises feitas (normalidade, homocedasticidade e dispersão (*outliers*) dos dados das amostras) foi possível partir para o teste paramétrico a fim de verificar se as médias entre as variáveis continham diferenças significativas podendo, assim, rejeitar ou aceitar as hipóteses nulas estabelecidas para as variáveis Erros, Quantidade de Acessos e Tempo. A Tabela 10 mostra o resultado do Teste T [21], um teste paramétrico para verificar a existência de diferenças significativas nas amostras.

Tabela 10 – Teste T

Variável	Igualdade das Variáveis	Significância (<i>2-tailed</i>)
Erros	Assumindo	1,000
	Não Assumindo	1,000
Quantidade de Acessos	Assumindo	0,118
	Não Assumindo	0,126
Tempo	Assumindo	0,985
	Não Assumindo	0,985

Para a variável **Erros**, o uso ou não da RDL não fez nenhuma diferença sob o ponto de vista estatístico. Nesse caso, não foi possível rejeitar a hipótese nula estabelecida para o experimento, de que o uso da RDL provoca um número de erros igual ao número de erros encontrados nos testes com os participantes que usaram técnica manual de instanciação de uma framework. Uma vez avaliada a variável **Erros** partiu-se para a avaliação referente à variável **quantidade de acessos**. Nessa variável há uma dispersão maior mas as médias, sob o ponto de vista estatístico, não contém diferenças significativas para que se possa concluir que houve diferença em relação ao uso da RDL e técnica manual. Ou seja, novamente não se conseguiu-se negar a hipótese nula e, portanto, o uso ou não da RDL não fez diferença em relação à **Quantidade de Acessos**. De forma similar à variável **Erros**, a variável **Tempo** também não apresentou variação entre o uso da RDL e de técnica manual.

4.4.5 Análise Qualitativa

Ao final da execução do experimento, cada participante respondia a uma espécie de questionário sob a forma de uma entrevista. As questões são mostradas no Apêndice A –. Os itens a seguir dizem respeito à compilação das respostas bem como impressões coletadas durante o processo de experimentação ou considerações espontâneas por parte dos participantes.

- Todos os participantes do experimento executam tarefas de instanciação de frameworks no dia-a-dia, ou seja, precisam de alguma forma utilizar os pontos de extensão de diferentes frameworks para gerar um produto final ou manter um existente.

- A média de tempo de leitura dos documentos (tanto dos requisitos como do próprio framework) para os que executaram o experimento com RDL ficou praticamente a mesma dos que executaram sem RDL, ficando apenas um pouco abaixo para os que executaram com RDL (RDL=10.7 minutos e Manual=10.5 minutos). Um participante comentou ao final do experimento que, pelo fato de estar apoiado em uma ferramenta (RDL), leu a documentação mais rápido do que deveria mas depois teve de reler para efetuar algumas correções de código.
- Sete dos oito participantes que executaram com RDL usaram da documentação do framework para o processo de instanciação. Destes sete, três recorreram ao diagrama para entender a hierarquia e relacionamento das classes.
- Dos que executaram com RDL, três tiveram de olhar o script RDL (Apêndice C –) durante processo de execução para entender o que estava sendo pedido naquele momento pois não tinham essa noção de forma clara na ferramenta Reuse Tool.
- Dos oito participantes que executaram o experimento com RDL, seis disseram ser favoráveis ao processo de geração de código automatizado mas que não abrem mão de ter uma boa documentação do framework. Os dois participantes contrários ao processo de geração de código automatizada disseram ter tido má experiência com ferramenta similar, principalmente em momentos de manutenção de código.
- Dos que executaram o experimento com RDL, 70% dos acessos a documentação foi feita durante o processo de codificação, sendo os restantes 30% durante o processo de instanciação com RDL.
- Dos oito participantes que executaram o experimento com RDL, três tiveram de recomençar o processo de instanciação com RDL por perderem a noção do que estava sendo feito em um determinado momento. Esse problema se deve ao fato de que a ferramenta Reuse Tool possui alguns diálogos confusos e não permite o retorno a um passo anterior.
- Dos oito participantes que executaram o experimento com RDL, metade preferiu não gerar os métodos de classe pela ferramenta Reuse Tool, preferindo deixar para a fase de codificação para corrigir os métodos pelo ambiente de desenvolvimento.
- A média em anos de experiência com desenvolvimento Java dos participantes ficou em seis anos e meio.

5 Conclusão

O trabalho aqui realizado trouxe para a comunidade científica dados relacionados a uma tarefa pertinente ao trabalho diário de arquitetos e engenheiros de software, que é a tarefa de instanciação de frameworks. Frameworks são importantes para o desenvolvimento de aplicações comerciais. Arquitetos e Engenheiros de Software usualmente investem na instanciação de frameworks, devendo para tal entender qual a estrutura do mesmo, seus pontos de variabilidade e suas características em geral e RDL foi proposta no sentido de contribuir com o processo de instanciação.

Entretanto, faz-se necessário a avaliação empírica deste tipos de proposta, para se ter a verdadeira noção de sua contribuição em um ambiente real. Por isso, esse trabalho agrega valor ao estado da arte ajustando a versão atual da linguagem RDL com a possibilidade do uso de interfaces e parâmetros nos métodos e também na verificação empírica do seu uso.

Porém, os resultados aqui apresentados não podem ser generalizados para um contexto diferente, como é o caso de se usar outras ferramentas de instanciação. Mesmo no contexto do trabalho aqui proposto, o que se pode afirmar é que RDL, para o tamanho do framework criado e do caso de uso proposto, não contribuiu com aumento de produtividade, diminuição de erros e/ou redução de acessos à documentação.

Mas conforme os dados apresentados na análise (seção 4.4.4) a produtividade dos participantes para a instanciação e codificação da aplicação final ficou muito similar uns aos outros, o que pode se inferir que RDL aproximou os tempos de desenvolvimento desses participantes. Ou seja, RDL, no contexto do trabalho, ao guiar os desenvolvedores a uma correta sequência de passos para a instanciação e conseqüente geração dos artefatos corretos utilizando os pontos de extensão do framework, ajudou a tornar os tempos de desenvolvimento mais uniformes, dando margem à interpretação de que poderia se utilizar abordagem similar para a estimativa de esforço.

Interpretações similares não puderam ser feitas para as outras variáveis (Tempo e Quantidade de Acessos). No caso de quantidade de acessos, a dispersão em RDL foi maior do que usando técnica manual, o que se conclui que o acesso à documentação do framework não diminui com o uso da RDL.

5.0.6 Trabalhos Futuros

É possível citar um conjunto de atividades que poderiam contribuir para um trabalho futuro envolvendo RDL. Por exemplo, poderia se replicar o experimento para verificar se o mesmo comportamento ocorre, em especial, com participantes não experientes ou com menos tempo de desenvolvimento do que os participantes do experimento realizado.

Ainda, aumentar a interatividade da linguagem RDL (seja através de um ambiente de desenvolvimento mais completo) buscando um apoio mais consistente a fim de reduzir o número de acessos a documentação, o número de passos ou até mesmo as dúvidas geradas pela ferramenta Reuse Tool. Funcionalidades adicionais de RDL poderiam ser utilizadas também como basear um experimento em um framework cujas extensões tem dependências entre os pontos de variabilidade. Nesse caso, RDL possui comandos para condicionar a extensão de um ponto à extensão de outro. Esses comandos não foram utilizados nesse experimento devido ao framework conter pontos de variabilidade mais simples.

Referências

- [1] OLIVEIRA, T. *Uma Abordagem Sistemática para a Instanciacao de Frameworks Orientados a Objetos*. 2001. 177 p. Tese de doutorado.
- [2] GAMMA, E. et al. *Design Patterns. elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Professional, 1995.
- [3] HIGHTOWER, R. *Jakarta-Struts Live*. Highlands Ranch, Colorado, USA: SourceBeat, LLC., 2004. ISBN 0974884308.
- [4] MICROSYSTEMS, S. *Design Patterns: Model-View-Controller*. [Http://java.sun.com/blueprints/patterns/MVC.html](http://java.sun.com/blueprints/patterns/MVC.html). Acesso em 30 de Novembro de 2008.
- [5] LUFT, C. P.; LUFT, L. *Mini Dicionario Luft*. [S.l.]: Editora Atica, 2002. ISBN 85-08-06988-X.
- [6] VEGAS, S. et al. Analysis of the influence of communication between researchers on experiment replication. In: *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*. New York, NY, USA: ACM, 2006. p. 28–37. ISBN 1-59593-218-6.
- [7] JURISTO. *Basics of Software Engineering Experimentation*. 101 Philip Drive, norwell, MA 02061, U.S.A: Kluwer Academic Publishers, 2001. ISBN 0-7923-7990-X.
- [8] BASILI, V. R. The role of experimentation in software engineering: past, current, and future. In: *ICSE '96: Proceedings of the 18th international conference on Software engineering*. Washington, DC, USA: IEEE Computer Society, 1996. p. 442–449. ISBN 0-8186-7246-3.
- [9] BASILI, V. R. Quantitative evaluation of software methodology. In: *Pan-Pacific Computer Conference*. Melbourne, Australia: [s.n.], 1985. p. 379–398.
- [10] KINNULA, A. *Software process engineering systems: models and industry cases*. [Http://herkules.oulu.fi/isbn9514265084/html/index.html](http://herkules.oulu.fi/isbn9514265084/html/index.html). Acesso em 30 de Novembro de 2008.
- [11] BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. In: *Encyclopedia of Software Engineering*. [S.l.]: Wiley, 1994. p. 528–532.
- [12] SJOBERG, D. I. K. et al. Conducting realistic experiments in software engineering. In: *ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2002. p. 17. ISBN 0-7695-1796-X.

- [13] TICHY, W. F. Should computer scientists experiment more? *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 31, n. 5, p. 32–40, 1998. ISSN 0018-9162.
- [14] OMG. *XML Metadata Interchange*. Omg.org/docs/formal/05-09-01.pdf. Acesso em 30 de Novembro de 2008.
- [15] PENCZEK, L.; MENDONCA, M.; OLIVEIRA, T. C. de. Systemizing aspect-oriented framework reuse with afr. In: *OOPSLA 06: Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 2006. p. 665–666. ISBN 1-59593-491-X.
- [16] FOWLER, M. *Inversion of Control Containers and the Dependency Injection pattern*. [Http://martinfowler.com/articles/injection.html](http://martinfowler.com/articles/injection.html). Acesso em 30 de Novembro de 2008.
- [17] MICROSYSTEMS, S. *Java*. Java.sun.com. Acesso em 30 de Novembro de 2008.
- [18] TIGRIS. *ArgoUML*. Argouml.tigris.org. Acesso em 30 de Novembro de 2008.
- [19] FOUNDATION, E. *Eclipse IDE*. Eclipse.org. Acesso em 30 de Novembro de 2008.
- [20] JUNIT.ORG. *JUnit*. Junit.org. Acesso em 30 de Novembro de 2008.
- [21] FIELD, A. *Discovering Statistics Using SPSS*. [S.l.]: SAGE Publications, 2005. ISBN 0761944524.

Apêndice A – Questionário para entrevista

1. Utilizou RDL?
2. Formação (Não Graduado - Graduado - Pós Graduado - Mestre - Doutor)
3. Experiência em Desenvolvimento de Software (em anos)
4. Grau de conhecimento de Java (Junior - Pleno - Senior)
5. Realiza instanciação de Frameworks no dia-a-dia?
6. Você acha que agrega valor a utilização de ferramentas que auxiliem processo de instanciação de frameworks?
7. O que acha que poderia ser melhorado (se utilizou RDL)?

Apêndice B – Caso de Uso para o Experimento

Os requisitos abaixo dizem respeito ao caso de uso codificado no experimento pelos participantes.

Fluxo normal dos eventos:

1. Usuário do sistema submete dados referentes ao artigo, conferência e submissão em si.
2. Sistema recebe dados da submissão contendo o artigo (Paper), um ou mais autores (Authors) e dados da conferência (Conference).
3. Sistema deverá fazer as seguintes validações
 - Se o paper é do tipo "ferramenta".
 - Se foi fornecido um título para o paper.
 - Se o número de páginas do Paper está no intervalo permitido, de 6 a 9.
 - Se o número de autores do Paper não ultrapassa o permitido para a conferência que é de no máximo 3 autores.
 - Só serão aceitos papers para sistemas operacionais Windows ou Linux.
 - Para qualquer validação que falhar, retornar false e informar a regra que falhou.
4. Sistema deve salvar todos os dados submetidos em formato XML.
5. Sistema deve enviar a pelo menos um autor uma notificação de que o Paper está sendo avaliado.
6. Sistema deve retornar true se todas as validações tiveram sucesso e false para qualquer validação que falhar.

Observação: Esse caso de uso é responsável apenas por construir objetos de negócio. Telas não fazem parte.

Apêndice C – Script RDL Utilizado no Experimento

```
instantiation cookbook MySubmission;

recipe main();

//declaration of objects
paperClass : class;
concretePaper : class;
conferenceClass : class;
conferenceExtClass : class;
submissionClass : class;
newAttribute : attribute;
newMethod : method;
submissionValIntf : interface;
submissionValImpl : class;
notificationIntf : interface;
notificationImpl : class;
universityClass : class;
authorClass : class;
abstractpersObject : class;
concretePersObject : class;

//NOTIFICATION
notificationIntf := get_interface("NotificationIntf");
notificationImpl := interface_implementation(notificationIntf,?);
loop
new_method(notificationImpl,?);
end_loop;
//END NOTIFICATION

//PERSISTENCE
abstractpersObject := get_class("AbstractPersistence");
concretePersObject := select_class_extension(abstractpersObject);
//asks reuser to add methods
loop
new_method(concretePersObject,?);
end_loop;
//END Persistence

//VALIDATION
submissionValIntf := get_interface("SubmissionValIntf");
```

```
submissionValImpl := interface_implementation(submissionValIntf,?);
loop
new_method(submissionValImpl,?);
end_loop;
//END VALIDATION

//PAPER
paperClass := get_class("Paper");
concretePaper := select_class_extension(paperClass);
//asks reuser to add methods
loop
new_method(concretePaper,?);
end_loop;
//END PAPER

end_recipe

end_cookbook
```


Apêndice D – Documentação do Framework

D.1 Introdução

O framework MySubmission é um framework escrito em tecnologia Java que provê funcionalidades básicas de manutenção de um sistema de conferências. Com MySubmission, é possível criar sistemas de conferências que suportam a submissão e manutenção de artigos (Papers), manutenção de autores (Authors) bem como todo o controle em torno da conferência. MySubmission ainda controla o ciclo de revisão de artigos.

D.2 Uso do Framework

A classe SubmissionFacade implementa o design pattern Façade e controla todo o fluxo de execução do framework. É a porta de entrada para todas as tarefas referentes ao processo de submissão de artigos. É nessa classe que se colocam as chamadas para as classes que irão implementar a persistência, a validação e notificações. Em um cenário típico, a classe SubmissionFacade recebe por parâmetro um objeto Submission preenchido com todos os dados da submissão e os objetos relacionados para realizar o seguinte fluxo:

- Obter uma referência a classe de validação e chamar o método responsável por validar os dados recebidos de acordo com as regras de negócio.
- Obter uma referência para um objeto de persistência e invocar o método responsável por persistir os dados em uma fonte persistente.
- Obter uma referência para o objeto de notificação e invocar o método responsável por enviar ao(s) autor(es) o status da revisão do artigo.

Cada método da classe SubmissionFacade deve retornar um valor booleano notificando o sucesso da operação (valor *true*) ou o não sucesso da operação escolhida (valor *false*).

D.2.1 Como obter referências de objetos para validações, persistência e notificações

Tanto para a validação, como para a persistência e notificações é necessário obter referências aos objetos que farão essas tarefas. A classe responsável por produzir instâncias de persistência, validação e notificação é a classe ObjectFactory que contém métodos concretos que retornam instâncias dessas classes. Não é necessário fazer nenhuma modificação nessa classe. Ela já contém métodos que retornam as instâncias desejadas. Para que ObjectFactory retorne a instância correta é necessário colocar o nome da classe que se deseja a instância no arquivo PaperSubmission.properties

Métodos de ObjectFactory

getInstance retorna uma referência a uma instância de *ObjectFactory*. Já ***getPersistenceObject*** retorna uma referência a uma implementação concreta de persistência. Exemplo de uso do código cliente:

```
XMLPersistence xml =
    ObjectFactory.getInstance().getPersistenceObject();
xml.insertPaper(paper);
```

ou

```
AbstractPersistence xml =
    ObjectFactory.getInstance().getPersistenceObject();
xml.insertPaper(paper);
```

getValidation Retorna uma referência a um objeto que representa uma instância da classe implementadora da interface *SubmissionValIntf*. Exemplo de uso do código cliente:

```
SubmissionValIntf val =
    ObjectFactory.getInstance().getValidation();
val.validateSubmission(sub);
```

getNotification Retorna uma referência a um objeto que representa uma instância da classe implementadora da interface *NotificationIntf*. Exemplo de uso código cliente:

```
NotificationIntf notif =
    ObjectFactory.getInstance().getNotification();
notif.sendRevision(submissionStatus);
```

D.2.2 Como implementar validações (Regras de Negócio)

A interface *SubmissionValIntf* precisa ser implementada para prover as validações específicas dos requisitos do sistema que usa o framework *MySubmission*. A classe *SubmissionFacade* precisa obter uma referência ao objeto que implementa a validação através da chamada do método *getValidation* da classe *ObjectFactory* e invocar o método necessário para a validação. O método de validação deve retornar um valor booleano representando o sucesso da validação (true) ou o não sucesso (false).

Para cada regra de validação que não for satisfeita é preciso utilizar um conjunto de mensagens de falha já pré-estabelecidos dentro da classe *PaperSubmissionConstants*. Para isso basta atribuir ao atributo *errorText* da classe *PaperSubmissionConstants* um dos valores que são constantes na própria classe e retornar false no método que implementou a validação.

D.2.3 Persistência de Dados

Uma classe que implementa a persistência também é obtida através da fábrica de objetos *ObjectFactory*, através do método *getPersistenceObject* (uma vez que se tenha informado dentro do arquivo de properties qual a classe que irá fazer a persistência da aplicação). Esse método retorna uma instância concreta que estende a classe *AbstractPersistence*, que contém métodos

abstratos que precisam ser implementados de acordo com a persistência escolhida. O framework *MySubmission* provê automaticamente a persistência em bancos relacionais ou em formato XML. Apenas coloque no arquivo *PaperSubmission.properties* a classe do framework que deseja que implemente a persistência e obtenha uma referência a ela pela classe *ObjectFactory*.

D.2.4 Notificações

Notificações precisam ser enviadas aos autores. Ao implementar a interface *NotificationIntf* o método *sendNotificationAuthor* precisa ser implementado. Esse método deve ser responsável por enviar notificação a um ou mais autores do estado da validação do Paper. O método *getNotification* da fábrica *ObjectFactory* é responsável por prover referência à instancia da implementação da interface *NotificationIntf* desde que seja colocado no arquivo *.properties* qual a classe concreta para fazer a notificação (a classe criada pelo próprio desenvolvedor e que implementa *NotificationIntf*). A classe *NotificationHelper* é uma classe que auxilia o processo de envio de notificações. De dentro de sua classe que constrói a notificação chame o método *send* da classe *NotificationHelper* passando um objeto *SubmissionStatus* devidamente configurado com o status da submissão do paper.

D.2.5 Submissão de Artigos

A classe *Submission* é uma classe que representa os dados da submissão (é um Data Transfer Object) e todo o relacionamento com os demais objetos. Para uma típica submissão de um artigo, os seguintes objetos devem ser criados e compostos:

Submission: Representa dados da submissão em si. Sua principal função é compor os demais objetos que fazem parte da submissão. Uma instância de *Submission* contém referência para uma instância de *Paper*. Já a instância de *Paper* contém referência para uma coleção de *Author*.

Paper: Classe abstrata que representa dados do artigo sendo submetido. As classes concretas são *PaperTool* e *PaperResearch*.

Author: Representa os dados do(s) autor(es) do artigo sendo submetido.

A Figura 19 mostra o relacionamento entre as classes que representam os dados.

A classe *Paper* é uma abstração para um artigo. Artigos podem ser relacionados a projetos de pesquisa ou ferramentas. Portanto, o sistema deve instanciar uma das classes que estendem da classe abstrata *Paper*.

D.2.6 Diagrama de Classes do Framework

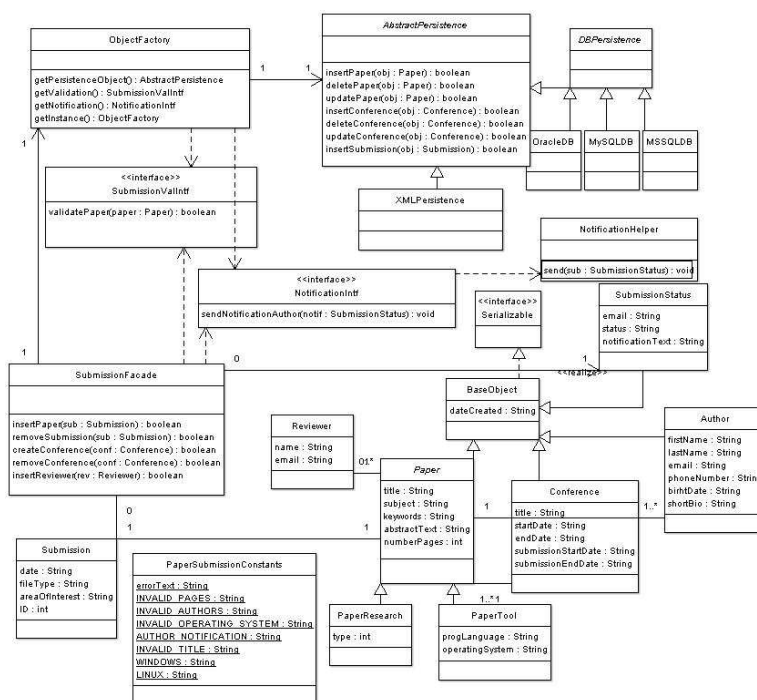


Figura 19 – Diagrama de Classes Framework MySubmission

Apêndice E – Arquivo de Especificação da Reuse Description Language

Tokens

```

white_space = (sp | ht | ff | line_terminator)*;
traditional_comment =
'/*' not_star+ '*' (not_star_not_slash not_star* '*'+)* '//';
end_of_line_comment = '//';
input_character* line_terminator?;

instantiation_cookbook = 'instantiation cookbook';
composition_cookbook = 'composition cookbook';
requires_instantiation = 'requires instantiation';
use = 'use';
end_cookbook = 'end_cookbook';

recipe = 'recipe';
main = 'main';
end_recipe = 'end_recipe';
out = 'out';

string = 'string';
number = 'number';
boolean = 'boolean';
reusable = 'reusable';
class_type = 'class';
interface_type = 'interface';
method = 'method';
attribute = 'attribute';
aspect = 'aspect';
pointcut = 'pointcut';
advice = 'advice';
joinpoint = 'joinpoint';

new_class = 'new_class';
new_method = 'new_method';
new_attribute = 'new_attribute';
new_class_inheritance = 'new_class_inheritance';
add_method_code = 'add_method_code';
get_class = 'get_class';
get_interface = 'get_interface';
get_method = 'get_method';
get_attribute = 'get_attribute';

```

```
new_aspect = 'new_aspect';
new_aspect_inheritance = 'new_aspect_inheritance';
new_pointcut = 'new_pointcut';
new_advice = 'new_advice';
add_advice_code = 'add_advice_code';
get_aspect = 'get_aspect';
get_pointcut = 'get_pointcut';
get_advice = 'get_advice';
element_choice = 'element_choice';
class_extension = 'class_extension';
interface_implementation = 'interface_implementation';
select_class_extension = 'select_class_extension';
method_extension = 'method_extension';
value_assignment = 'value_assignment';
value_selection = 'value_selection';
aspect_extension = 'aspect_extension';
select_aspect_extension = 'select_aspect_extension';
pointcut_extension = 'pointcut_extension';
add_joinpoint = 'add_joinpoint';
inheritance_introduction = 'inheritance_introduction';
method_introduction = 'method_introduction';
attribute_introduction = 'attribute_introduction';
cmd_and = '#';
cmd_or = 'o';
cmd_xor = 'xo';
while = 'while';
do = 'do';
end_while = 'end_while';
loop = 'loop';
end_loop = 'end_loop';
if = 'if';
then = 'then';
else = 'else';
end_if = 'end_if';
user_interaction = '?';

l_parenthese = '(';
r_parenthese = ')';
l_bracket = '[';
r_bracket = ']';
semicolon = ';';
colon = ':';
comma = ',';

true = 'true';
false = 'false';
null = 'null';

assign = ':=';

plus = '+';
```

```

minus = '-';
star = '*';
div = '/';
mod = '%';
lt = '<';
gt = '>';
lteq = '<=';
gteq = '>=';
eq = '=';
neq = '<>';
    not = 'not';
and = 'and';
or = 'or';
xor = 'xor';
    lcfirst = 'lcfirst';
    ucfirst = 'ucfirst';

number_literal = digit+ | digit+ '.' digit+;
string_literal = '"' string_character* '"';

identifier = letter letter_or_digit*;

```

Ignored Tokens

```

white_space,
traditional_comment,
end_of_line_comment;

```

Productions

```

goal =
    compilation_unit;

literal =
    {number_literal}
        number_literal |

    {boolean_literal}
        boolean_literal |

    {string_literal}
        string_literal |

    {null_literal}
        null;

boolean_literal =
    {true} true |
    {false} false;

type =

```

```

{string}
    string |
{number}
    number |
{boolean}
    boolean |
{reusable}
    reusable |
{interface_type}
    interface_type |
{class_type}
    class_type |
{method}
    method |
{attribute}
    attribute |
{aspect}
    aspect |
{pointcut}
    pointcut |
{advice}
    advice |
{joinpoint}
    joinpoint |
{array}
    type l_bracket r_bracket;

name = identifier;

compilation_unit =
    {instantiation}
        instantiation_cookbook_declaration |
    {composition}
        composition_cookbook_declaration;

instantiation_cookbook_declaration =
    instantiation_cookbook name uses_declaration?
semicolon cookbook_body end_cookbook;

composition_cookbook_declaration =
    composition_cookbook name requires_instantiation?
uses_declaration? semicolon cookbook_body end_cookbook;

uses_declaration = use library_list;

library_list =
    {library}
        name |
    {library_list}

```



```

        name comma library_list;

cookbook_body = recipe_main_declaration
recipe_declaration*;

recipe_main_declaration =
    recipe main parameter_declaration?
semicolon command_list end_recipe;

recipe_declaration =
    recipe name parameter_declaration?
semicolon command_list end_recipe;

parameter_declaration = l_parenthese parameter_list? r_parenthese;

parameter_list =
    {parameter}
    parameter |

    {parameter_list}
    parameter comma parameter_list;

parameter =
    out? name colon type;

command_list =
    {command}
    cmd_or_seq_cmd semicolon |
    {command_list}
    cmd_or_seq_cmd semicolon command_list;

cmd_or_seq_cmd =
    {command}
    command |
    {sequence_or_command}
    command1 cmd_or command2 |
    {sequence_xor_command}
    command1 cmd_xor command2 |
    {sequence_and_command}
    command1 cmd_and command2;

command1 = command;
command2 = command;

command =
    {variable_declaration}
    variable_declaration |
    {assignment}
    assignment |
    {expression_command}

```

```

        expression_command |
    {language_command}
        language_command |
    {recipe_call}
        recipe_call;

variable_declaration = name colon type;

assignment = name assign expression;

recipe_call = name argument_declaration?;

argument_declaration = l_parenthese
argument_list? r_parenthese;

argument_list =
    {argument}
        argument |

    {argument_list}
        argument comma argument_list;

argument = expression;

language_command =
{while}
while expression do command_list end_while |
{loop}
loop command_list end_loop |
{if_then}
if expression then_clause else_clause? end_if |
{new_class_inheritance}
new_class_inheritance l_parenthese
parameter1 parameter2 r_parenthese |
{add_method_code}
add_method_code l_parenthese parameter1
parameter2 r_parenthese |
{new_aspect_inheritance}
new_aspect_inheritance l_parenthese
parameter1 parameter2 r_parenthese |
{add_advice_code}
add_advice_code l_parenthese parameter1
parameter2 r_parenthese |
{value_assignment}
value_assignment l_parenthese parameter1
parameter2 r_parenthese |
{value_selection}
value_selection l_parenthese parameter1
parameter2 r_parenthese |
{add_joinpoint}
add_joinpoint l_parenthese parameter1

```

```

parameter2 r_parenthese |
{inheritance_introduction}
inheritance_introduction l_parenthese
  parameter1 parameter2 parameter3 r_parenthese;

then_clause = then command_list;

else_clause = else command_list;

parameter1 = expression;

parameter2 = comma expression;

parameter3 = comma expression;

expression_command =
{user_interaction}
user_interaction |
{new_class}
new_class l_parenthese parameter1 r_parenthese |
{new_method}
new_method l_parenthese parameter1 parameter2 r_parenthese |
{new_attribute}
new_attribute l_parenthese parameter1 parameter2 r_parenthese |
{get_class}
get_class l_parenthese parameter1 r_parenthese |
{get_interface}
get_interface l_parenthese parameter1 r_parenthese |
{get_method}
get_method l_parenthese parameter1 parameter2 r_parenthese |
{get_attribute}
get_attribute l_parenthese parameter1 parameter2 r_parenthese |
{new_aspect}
new_aspect l_parenthese parameter1 r_parenthese |
{new_pointcut}
new_pointcut l_parenthese parameter1 parameter2 r_parenthese |
{new_advice}
new_advice l_parenthese parameter1 r_parenthese |
{get_aspect}
get_aspect l_parenthese parameter1 r_parenthese |
{get_pointcut}
get_pointcut l_parenthese parameter1 parameter2 r_parenthese |
{get_advice}
get_advice l_parenthese parameter1 r_parenthese |
{element_choice}
element_choice l_parenthese parameter1 r_parenthese |
{class_extension}
class_extension l_parenthese parameter1 parameter2
r_parenthese |
{interface_implementation}
interface_implementation l_parenthese parameter1

```

```

parameter2 r_parenthese |
{select_class_extension}
select_class_extension l_parenthese parameter1 r_parenthese |
{method_extension}
method_extension l_parenthese parameter1 parameter2
parameter3 r_parenthese |
{aspect_extension}
aspect_extension l_parenthese parameter1 parameter2
r_parenthese |
{select_aspect_extension}
select_aspect_extension l_parenthese parameter1
r_parenthese |
{pointcut_extension}
pointcut_extension l_parenthese parameter1 parameter2
parameter3 r_parenthese |
{method_introduction}
method_introduction l_parenthese parameter1 parameter2
parameter3 r_parenthese |
{attribute_introduction}
attribute_introduction l_parenthese parameter1 parameter2
parameter3 r_parenthese;

```

```

expression =
{l_parenthese}
l_parenthese expression r_parenthese |
{conditional_or_expression}
conditional_or_expression;

```

```

primary =
{name}
name |
{literal}
literal |
{command}
expression_command ;

```

```

unary_expression =
{not}
not unary_expression |
{lcfirst}
lcfirst unary_expression |
{ucfirst}
ucfirst unary_expression |
{primary}
primary;

```

```

multiplicative_expression =
{unary_expression}
unary_expression |
{star}
multiplicative_expression star unary_expression |

```

```
{div}
    multiplicative_expression div unary_expression |
{mod}
    multiplicative_expression mod unary_expression;

additive_expression =
    {multiplicative_expression}
        multiplicative_expression |
    {plus}
        additive_expression plus multiplicative_expression |
    {minus}
        additive_expression minus multiplicative_expression;

relational_expression =
    {additive_expression}
        additive_expression |
    {lt}
        relational_expression lt additive_expression |
    {gt}
        relational_expression gt additive_expression |
    {lteq}
        relational_expression lteq additive_expression |
    {gteq}
        relational_expression gteq additive_expression;

equality_expression =
    {relational_expression}
        relational_expression |
    {eq}
        equality_expression eq relational_expression |
    {neq}
        equality_expression neq relational_expression;

conditional_and_expression =
    {equality_expression}
        equality_expression |
    {conditional_and_expression}
        conditional_and_expression and equality_expression;

conditional_or_expression =
    {conditional_and_expression}
        conditional_and_expression |
    {conditional_or_expression}
        conditional_or_expression or conditional_and_expression |
    {conditional_xor_expression}
        conditional_or_expression xor conditional_and_expression;
```