UNIVERSIDADE FEDERAL DE MATO GROSSO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMAS DE INFORMAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO
UTILIZAÇÃO DE PADRÕES DE DESENVOLVIMENTO PARA REESTRUTURAÇÃO DE SISTEMAS

MARCOS FELIPE SANTOS DE OLIVEIRA

CUIABÁ – MT
2014

UNIVERSIDADE FEDERAL DE MATO GROSSO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMAS DE INFORMAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO
UTILIZAÇÃO DE PADRÕES DE DESENVOLVIMENTO PARA REESTRUTURAÇÃO DE SISTEMAS

MARCOS FELIPE SANTOS DE OLIVEIRA

Relatório apresentado Instituto de Computação da Universidade Federal de Mato Grosso, para obtenção do título de Bacharel em Sistemas de Informação.

CUIABÁ – MT
2014

UNIVERSIDADE FEDERAL DE MATO GROSSO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

MARCOS FELIPE SANTOS DE OLIVEIRA

Relatório de Estágio Supervisionado apresentado à Coordenação do Curso de Sistemas de Informação como uma das exigências para obtenção do título de Bacharel em Sistemas de Informação da Universidade Federal de Mato Grosso

Aprovado por:

Prof. MSc. Nilton Hideki Takagi

Instituto de Computação

(COORDENADOR DE ESTÁGIOS)

Prof. MSc. Allan Gonçalves de Oliveira

Instituto de Computação

(ORIENTADOR)

MSc. Marco Antonio dos Anjos

(SUPERVISOR)

DEDICATÓRIA

A Deus graças pela minha vida.

À minha família pela minha criação e o apoio dado até hoje.

A meus professores por seus ensinamentos.

AGRADECIMENTOS

Agradeço a todos meus familiares, amigos e professores que me apoiaram durante todos os anos dessa graduação.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

CASE	Computer Aided Software Engineering
IDE	Integrated Development Environment
MVC	Model-View-Controller
TCE-MT	Tribunal de Contas do Estado de Mato Grosso

RESUMO

Este trabalho é um relatório de estágio supervisionado realizado para a obtenção do título de bacharel em Sistemas de Informação na Universidade Federal de Mato Grosso. O estágio foi realizado no Tribunal de Contas do Estado de Mato Grosso dentro da Coordenadoria de Tecnologia da Informação e teve como área de atuação a de desenvolvimento. O principal objetivo do trabalho foi facilitar a manutenção e o teste de um sistema já em produção. Foi aproveitado o momento de uma grande demanda de novas funcionalidades para efetuar uma reestruturação em todo o sistema aplicando padrões de desenvolvimento. O uso de padrões de desenvolvimento se mostrou importante para facilitar a manutenção do sistema. Além disso, tornou menos complicado o trabalho em equipe e a adesão de novos desenvolvedores ao projeto. O sistema utilizado como base para o relatório foi desenvolvido em uma ferramenta case, o GeneXus. Por razão das particularidades do sistema e da ferramenta utilizada houve a necessidade de se adaptar um padrão de desenvolvimento para atender as demandas. O padrão utilizado como base foi o MVC, amplamente utilizado na linguagem Java.

Palavras-chave: Padrões de Desenvolvimento, Ferramentas Case, MVC, GeneXus

INTRODUÇÃO

É muito comum hoje em dia desenvolver sistemas que acabam extrapolando seu escopo inicial, sendo comum nesses casos o uso de uma arquitetura inadequada para o tamanho do sistema final. Muitas vezes se faz necessário a reestruturação do sistema para facilitar a manutenção e o desenvolvimento do mesmo.

A manutenção do sistema utilizado como base para o relatório tornava-se cada vez mais complexa com a adição de funcionalidades. Com o surgimento de uma grande quantidade de novas demandas tornou-se visível a necessidade de uma mudança.

O uso de padrões de desenvolvimento simplifica o processo de desenvolvimento e manutenção, além de dar apoio à evolução do sistema, possibilitar o reuso de código, entre outros benefícios.

Por se tratar de conceitos e boas práticas de desenvolvimento amplamente testadas, o uso de padrões de desenvolvimento garante determinadas vantagens quando aplicado da maneira correta.

O objetivo desse projeto é analisar as vantagens do uso de padrões de desenvolvimento e os resultados obtidos.

O relatório apresenta brevemente as tecnologias utilizadas no estudo de caso e se aprofunda nos padrões de desenvolvimento com foco naqueles utilizados como base para reestruturar o sistema abordado.

1. REVISÃO DE LITERATURA

Neste capítulo são abordados os conceitos teóricos utilizados no projeto realizado durante o estágio supervisionado. O principal conceito utilizado foi o de padrões de desenvolvimento. Pelo fato de os padrões terem sido aplicados em um projeto apoiado por uma ferramenta CASE, o conceito de ferramentas CASE também será abordado neste capítulo.

1.1 PADRÕES DE DESENVOLVIMENTO

Christopher Alexander, arquiteto, matemático e urbanista austríaco, foi um dos primeiros a citar "padrões" em seus trabalhos na área de arquitetura e engenharia civil. Segundo ele, cada padrão descreve um problema que ocorre repetidamente em nosso meio, e então descreve o núcleo da solução para aquele problema, de uma forma que você possa utilizar essa solução um milhão de vezes, sem nunca ter feito da mesma forma duas vezes. Apesar de Alexander ter falado sobre construções e cidades, o que ele disse também é realidade quando se fala em padrões de desenvolvimento. Segundo os autores GAMMA, HELM, JOHNSON e VLISSIDES (1994), um padrão de desenvolvimento possui quatro elementos essenciais :

1. **Nome do padrão:** é o nome utilizado para descrever um problema de projeto, suas soluções e consequências em uma palavra ou duas.
2. **Problema:** é a descrição do problema e seu contexto.
3. **Solução:** é uma descrição abstrata de como um arranjo genérico de elementos resolvem o problema. A solução não descreve uma implementação particular e concreta para o problema, pois um padrão é como um modelo que pode ser aplicado em diversas situações diferentes.
4. **Consequências:** são os resultados de se aplicar o padrão. Embora as consequências sejam frequentemente desconsideradas ao se descrever decisões de projeto, elas são críticas para avaliar alternativas de projeto e para entender os custos e benefícios de se aplicar o padrão.

Padrões de desenvolvimento podem acelerar o processo de desenvolvimento fornecendo paradigmas de desenvolvimento testados e provados. O desenvolvimento efetivo de software exige considerar problemas que podem não se apresentar até o fim da implementação. O reuso de padrões de desenvolvimento ajuda a prevenir problemas sutis que podem causar problemas maiores e melhora a legibilidade para desenvolvedores e arquitetos familiarizados com os padrões.

Frequentemente, as pessoas entendem apenas como aplicar certos tipos técnicas de desenvolvimento de software a certos problemas. Essas técnicas são difíceis de se aplicar a uma gama maior de problemas. Padrões de desenvolvimento fornecem soluções genéricas, documentadas em um formato que não requer especificidades atreladas a um problema particular.

Além disso, segundo SHVETS, padrões permitem a desenvolvedores se comunicar utilizando nomes bem conhecidos e entendidos para interações de software. Padrões de desenvolvimento comuns podem ser melhorados com o tempo, tornando-os mais robustos que implementações específicas.

1.1.1 CLASSIFICAÇÃO DOS PADRÕES DE DESENVOLVIMENTO

Os padrões de desenvolvimento podem ser classificados em, segundo SHVETS:

- **Padrões de criação (*Creational*):** São padrões sobre instanciação de classes. Podem se subdivididos em dois grupos: padrões de criação de classes e padrões de criação de objetos. Enquanto os padrões de criação de classe usam herança para o processo de instanciação os padrões de criação de objeto utilizam delegação.
- **Padrões de estrutura (*Structural*):** São padrões sobre composição de classes e objetos. Utilizam herança para compor interfaces e definem formas de compor objetos para obter novas funcionalidades.
- **Padrões de comportamento (*Behavioral*):** São padrões especificamente relacionados com a comunicação entre objetos.

A utilização de padrões de desenvolvimento possui diversas vantagens já citadas anteriormente. Porém, os padrões de desenvolvimento não tornam o projeto isento de falhas, pois mesmo com seu uso a qualidade do projeto só é garantida com uma boa implementação. Caso seja mal aplicado o padrão pode diminuir a compreensão do projeto e aumentar a quantidade de código.

Para avaliar a viabilidade do uso dos padrões de desenvolvimento é preciso bom senso. O problema, a solução e a consequência do uso do padrão devem ser bem estudados para definir se determinado padrão será benéfico para o projeto em questão. Ter uma visão global do projeto e seu funcionamento facilita essa tarefa.

1.2 FERRAMENTAS CASE

A complexidade dos sistemas atuais trouxe a necessidade de uma sistemática para a modelagem de problemas do mundo real em softwares. As ferramentas de apoio ao desenvolvimento foram um grande avanço para diminuir os esforços humanos, agilizar o processo de desenvolvimento e garantir a qualidade do produto final.

Os sistemas computacionais são imprescindíveis nos dias de hoje. Nas suas mais diversas áreas de atuação eles se tornam cada vez mais adaptados as necessidades dos usuários e possuem funcionalidades cada vez mais específicas.

O processo de desenvolvimento de softwares sofreu diversos aprimoramentos. Notações, formalismos, modelos e metodologias surgiram para dar suporte ao processo, reduzindo custos, aumentando a produtividade e melhorando a qualidade do produto final.

FARIAS (2001) cita, “A sofisticação dos métodos de desenvolvimento acarreta o aumento da complexidade da administração do processo. Tal como uma fábrica, a produção de software demanda de ferramentas de apoio capazes de eliminar problemas usuais decorrentes das limitações humanas”.

Com o aumento da demanda, surgiram as ferramentas CASE para atender as diversas fases do processo de desenvolvimento de software. KURBEL e SCHNIEDER (1994) destacam que, embora cada ferramenta possa representar um diferencial, os maiores benefícios do uso das ferramentas surge com a integração. O uso de uma ferramenta para cada fase do processo necessita que a correspondência entre o produto de cada fase seja garantida por engenheiros, tornado o processo suscetível a erros humanos. Com sistemas integrados, essas problemáticas seriam tratadas pela ferramenta, eliminando erros e diminuindo o trabalho dos desenvolvedores.

Computer Aided Software Engineering envolve o uso de computadores para apoiar o processo de desenvolvimento de software. Essa simples visão caracteriza o CASE desde seu desenvolvimento no começo dos anos 70. No entanto, o CASE se tornou muito mais que ferramentas automatizadas para dar suporte ao processo de Engenharia de software. Hoje em dia o CASE evoluiu para uma abordagem completa para a concepção e produção de softwares, como evidenciado pela grande variedade de ferramentas disponíveis que contribuem para o ambiente CASE .

1.2.1 CLASSIFICAÇÃO DAS FERRAMENTAS CASE

O grande número de ferramentas disponíveis torna a tarefa de classificá-las complexa. MANLEY (1990) destaca que, graças a grande diversidade de ferramentas é compreensível que, saber o que cada ferramenta faz e a comparar com as demais se torna uma tarefa muito mais difícil. Por isso, existem algumas áreas distintas em que as ferramentas podem ser categorizadas:

- Cobertura do ciclo de vida.
    - *Front End* ou *Upper* CASE: apoia as fases de planejamento, análise e projeto do programa ou aplicação.
    - *Back End* ou *Lower* CASE: dão apoio à parte física, isto é, a codificação testes e manutenção da aplicação.
    - I-CASE ou *Integrated* CASE: classifica os produtos que cobrem todo o ciclo de vida do software, desde os requisitos do sistema até o controle final da qualidade.

- Nível de Integração.
  - *Tools*: suportam tarefas específicas no ciclo de vida do software.
  - *Workbenches*: combinam duas ou mais Tools focando uma parte específica do ciclo de vida do softwate.
  - *Enviroments*: combinam duas ou mais *Tools* ou *Workbenches* dando suporte a todo o ciclo de vida do software.
- Funcionalidade.
  - A tabela 1 mostra a classificação segundo a funcionalidade das ferramentas.

1.2.2 MÉTRICAS

Utilizado como sistema de medição, o PEI (*Production Efficiency Index*) aplica-se genericamente a qualquer processo de produção. Para os processos de desenvolvimento de software existe uma variação do PEI que é utilizada, o SPEI (*Software Production Efficiency Index*) é necessário para avaliar o impacto da tecnologia CASE na eficiência da produção, independentemente da complexidade do produto, experiência da equipe envolvida, características do ambiente de desenvolvimento, etc (FARIAS, 2001). Este método pode ser utilizado para avaliar os seguintes aspectos:

- Impacto das ferramentas CASE no processo de produção de forma detalhada: design, codificação e teste.
- Impacto de outros fatores no processo: treinamento, melhorias do processo, melhorias de hardware.
- Identificar fatores que impactam na eficiência de produção, servindo de base para o diagnóstico da produtividade e novas tomadas de decisões na organização.
- Prover formas de medições em função de tempo: estimar produtividade, custos e tempo no desenvolvimento do software.

Tabela 1: Classificação por funcionalidade (Farias, 2001)

Tipo	Descrição	Exemplos
Gerenciamento	Servem para representar os elementos de um processo e seus relacionamentos. Úteis para entender o processo e atividades necessárias a sua execução.	Ferramentas de estimação de custos.
Edição	Úteis para codificação, modelagem gráfica e documentação.	Editores de textos, editores de diagramas.
Gerenciamento de Configuração	Ajudam nas tarefas de identificação, controle de versão, controle de mudança, auditoria, etc.	Sistemas de controle de versão e sistemas de gerenciamento de mudança.
Prototipagem	Ajudam na criação rápida de protótipos. Suporte a definição de layouts de telas, integração de telas com o design dos dados, geração de código fonte.	Geradores de interfaces gráficas, linguagens de alto nível.
Suporte a programação	Permitem o desenvolvedor construir o software em uma linguagem mais inteligível a humanos.	Compiladores, interpretadores, editores.
Análise de programas	Analisam tanto o código fonte quanto interagem com o sistema em execução. Úteis para gerar casos de testes e inspecionar o fluxo do programa.	Analisadores estáticos e dinâmicos.
Teste	Usadas para controlar e coordenar os testes de software.	Geradores de dados para teste, comparadores de arquivos.
Depuração	Servem para inspecionar o código fonte em execução, tornando possível uma observação rigorosa da execução do sistema.	Sistemas interativos de depuração.
Documentação	Ajudam na produção e publicação dos documentos.	Editores de imagens, processadores de textos.
Reengenharia	Servem para engenharia reversa, reestruturação de código.	Sistemas de reestruturação de programas.

2. MATERIAIS, TÉCNICAS E MÉTODOS

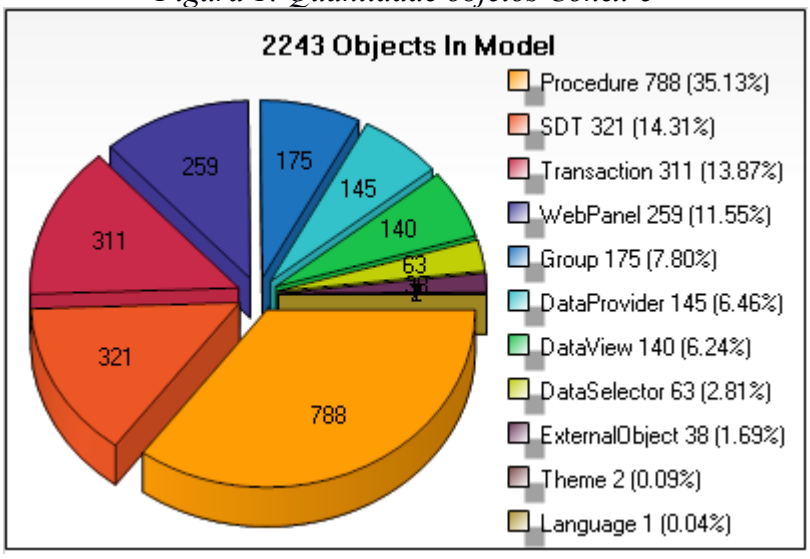
Neste capítulo é abordada a aplicação dos padrões de desenvolvimento no sistema acompanhado durante o estágio supervisionado. O contexto do sistema, suas tecnologias e arquitetura serão apresentados na introdução, em sequência o padrão de desenvolvimento utilizado será explicado mais detalhadamente.

2.1. INTRODUÇÃO

Considerando o ambiente de um órgão público estadual no Brasil, muitas vezes questões politicas acabam interferindo na escolha dos sistemas e tecnologias utilizados. Nesse contexto é muito comum ter um esforço extra para atender as demandas com o que se tem disponível. O sistema utilizado como base para o estágio supervisionado foi implementado em GeneXus Ev2, é atendido por alguns serviços implementados em Java 1.6 e roda sobre o banco de dados Oracle 11g.

O sistema em questão, chamado Conex-e, surgiu com o propósito de automatizar o processo de geração de relatórios técnicos do Tribunal de Contas do Estado de Mato Grosso (TCE-MT) e dessa forma facilitar e agilizar o trabalho dos servidores lotados nas secretarias de Controle Externo. Muitas informações, cálculos e levantamentos feitos manualmente, são automatizados e gerados pelo Conex-e por meio da integração dos sistemas técnicos do TCE-MT. A Figura 1 demonstra a quantidade de objetos do sistema Conex-e.

Figura 1: Quantidade objetos Conex-e



Durante o período do estágio supervisionado diversas demandas de novas funcionalidades para o sistema foram apresentadas. Nesse meio tempo, algumas dificuldades foram notadas e foi previsto que com o crescimento do sistema elas poderiam se tornar grandes problemas. As dificuldades se resumiam ao teste, que se tornava cada vez mais complexo e demorado, e a manutenção, que devido a arquitetura utilizada se tornaria mais custosa com o sistema cada vez maior. Ainda era previsto, com base nessas dificuldades, que caso houvesse troca ou adesão à equipe seria trabalhoso familiarizar novos integrantes ao sistema.

Para evitar os problemas com a adição das novas funcionalidades foi necessário repensar a arquitetura do sistema. Para tanto, desenvolvedores, gerentes e consultores se reuniram e duas propostas principais de reestruturação foram apresentadas.

Proposta 1

O sistema antes da reestruturação era basicamente todo implementado em GeneXus para web com algumas rotinas específicas, como geração de PDF, implementadas em Java e utilizadas como serviço pelo GeneXus. A primeira proposta apresentada consistia em transferir todas as regras de negócio do GeneXus para o Java e manter apenas a interface gráfica como responsabilidade do GeneXus.

As vantagens dessa proposta eram:

- Agilizar e facilitar os testes através do uso do JUnity, um framework que facilita a criação de código para a automação de testes com apresentação dos resultados. Com ele é possível verificar se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas, e executar baterias de testes.
- Facilitar a manutenção do sistema, aplicando padrões de desenvolvimento durante a migração de código do GeneXus para o Java.

As desvantagens eram:

- A necessidade de reescrever quase todo o sistema em um curto período de tempo, considerando o fato de que por se tratar de uma ferramenta de alto

nível o GeneXus tem linguagem mais abstrata que o Java tornando assim seu código mais enxuto. Isso afetaria de forma considerável o cronograma das novas implementações.

- O fato de o número de desenvolvedores GeneXus ser maior que o de desenvolvedores Java na equipe, no período.

Proposta 2

Considerando que o sistema foi implementado quase em sua totalidade em GeneXus a segunda proposta consistia em reorganizar a estrutura do sistema, aproveitando o que já havia sido implementado, para atender as novas demandas sem tornar mais complexos o teste e a manutenção do sistema.

As vantagens dessa proposta eram:

- A maior agilidade de implementação fornecida pela ferramenta GeneXus.
- Facilitar a manutenção do sistema, aplicando padrões de desenvolvimento numa reestruturação do sistema durante a implementação das novas funcionalidades. Dessa forma a reestruturação seria gradual de acordo com o andamento das novas implementações e a utilização de código já existente. Além disso, cronograma das novas implementações seria minimamente afetado.

A desvantagem era:

- A falta de uma ferramenta específica de testes e a dificuldade de se trabalhar com a função de debug da ferramenta GeneXus.

Proposta final

Após diversas reuniões e aprimoramento das propostas ficou definido que a segunda seria a mais viável, considerando o tempo disponível. Porém ainda era necessário definir uma solução para a deficiência da proposta escolhida, o teste. Para tanto, foi necessário buscar uma forma efetiva de se facilitar e agilizar os testes do sistema através da ferramenta GeneXus.

2.2. GENEXUS

GeneXus é uma ferramenta CASE criada pela empresa uruguaia ARTech, baseada em conhecimento e orientada ao desenvolvimento de aplicações corporativas para web, desktop e recentemente dispositivos mobile. É uma das primeiras ferramentas inteligentes com capacidade de criar e manter, de forma automática, aplicações multiplataforma que se adaptam às mudanças de negócio (AQUINO).

O GeneXus tem como uma de suas principais vantagens a agilidade no desenvolvimento. O desenvolvedor esboça suas aplicações em alto nível, principalmente em linguagem declarativa, então a ferramenta gera código para múltiplas plataformas. Por utilizar uma linguagem de alto nível proprietária, isto proporciona a ferramenta a capacidade de traduzir estas instruções para várias outras linguagens.

Além disso, inclui um módulo de normalização, que cria e mantém uma estrutura de banco de dados otimizada automaticamente, baseada no modelo de dados não normalizado definido pelo usuário através de uma linguagem declarativa e uma linguagem procedural simples e poderosa. Através da ferramenta é rápido e fácil criar aplicações de diversos níveis de complexidade como, por exemplo, aplicativos ERP, sites, portais, aplicativos para celulares.

Apesar de todas as suas qualidades o GeneXus não é perfeito. Uma ferramenta por si só não é o suficiente para garantir a criação e manutenção de grandes sistemas de forma eficaz. Para tanto é necessário avaliar o custo, o suporte, a documentação e a comunidade de usuários da ferramenta. É ai que aparecem os problemas do GeneXus. O custo pode ser compensado com a produtividade fornecida, porém o suporte muitas vezes é falho, a documentação não é suficiente e a comunidade de usuários ainda é pequena. O resultado disso é que no meio de tantas funcionalidades existem algumas, como o debug, que acabam sendo descartadas por falta de documentação. Na maioria das vezes é muito difícil se encontrar publicações na internet sobre o GeneXus, suas funcionalidades e usos.

2.3. A REESTRUTURAÇÃO DO SISTEMA

Durante o processo de pesquisa de tecnologias e ferramentas para a reestruturação do sistema uma ferramenta de testes unitários foi elencada para a segunda proposta, sendo equivalente a JUnity da primeira proposta. Entretanto, a ferramenta, chamada GXtest, foi descartada pois sua documentação e suporte eram falhos e seu uso foi desaconselhado por pessoas que a haviam utilizado. Com isso, a utilização do GeneXus no Conex-e possuía dois desfalques, uma ferramenta de testes unitários e a função de debug.

Ambas falhas foram devidamente corrigidas com práticas de desenvolvimento. Para o teste do sistema, seriam criadas rotinas dentro do próprio sistema para automatizar o que fosse possível. Já para debugar o código durante a implementação foi mantido o uso de logs para rastreio de variáveis, tornando possível o acompanhamento da execução do sistema.

Mesmo não sendo a solução ideal, essas práticas de desenvolvimento já conhecidas e utilizadas podem facilitar e agilizar os testes do sistema. Com as falhas da proposta de reestruturação final solucionadas foi possível dar início a reestruturação do sistema em paralelo a implementação das novas funcionalidades do sistema.

Graças ao fato de que muitas das novas funcionalidades eram diretamente ligadas ao que já havia sido implementado não houve grande dificuldade para reestruturar o sistema durante as novas implementações. Sempre que uma nova funcionalidade fazia referência a algo já implementado o código antigo era revisto e reescrito na nova arquitetura definida, caso fosse necessário. Para se definir uma nova arquitetura que atendesse os requisitos do sistema em crescimento foram utilizados padrões de desenvolvimento.

2.4. APLICANDO PADRÕES DE DESENVOLVIMENTO

Uma arquitetura escalável e bem construída torna possível o crescimento de um sistema sem tornar complexa a sua manutenção. Porém, um sistema pode crescer muito além do que foi previsto em sua elaboração, o que torna a criação de sua arquitetura uma tarefa complexa. Uma forma de criar uma boa estrutura para um sistema é utilizar paradigmas de desenvolvimento testados e provados. Para tanto, são utilizados os padrões de desenvolvimento.

No caso do Conex-e a estrutura do código utilizada já estava causando problemas de manutenção antes mesmo das novas demandas. Havia, por exemplo, muito código repetido por todo o sistema, que ao ser alterado em um ponto, também precisava ser alterado nos outros, causando retrabalho. Se um código repetido fosse alterado por mais de um desenvolvedor ao mesmo tempo, ambos teriam que juntar os códigos manualmente e reescrever o mesmo código nas outras partes do sistema onde ele era utilizado.

Para evitar problemas futuros com o sistema, alguma mudança deveria ser feita. Reescrever o sistema do zero não era viável pois ele já possuía um tamanho substancial e não havia tempo hábil para refazê-lo com as novas funcionalidades sem atrasar o cronograma. Foi então que surgiu a ideia de reestruturar a arquitetura do sistema através do uso de padrões de desenvolvimento.

O sistema necessitava de uma arquitetura modular, que separasse de uma forma prática a interface das regras de negócio, para que ambas pudessem ser trabalhadas ao mesmo tempo por desenvolvedores diferentes. Além disso, também era preciso acabar com códigos replicados, centralizando o código replicado e reutilizando-o onde fosse preciso. Uma melhor organização do código também era bem-vinda, para evitar o retrabalho e facilitar a busca dentro do código já escrito.

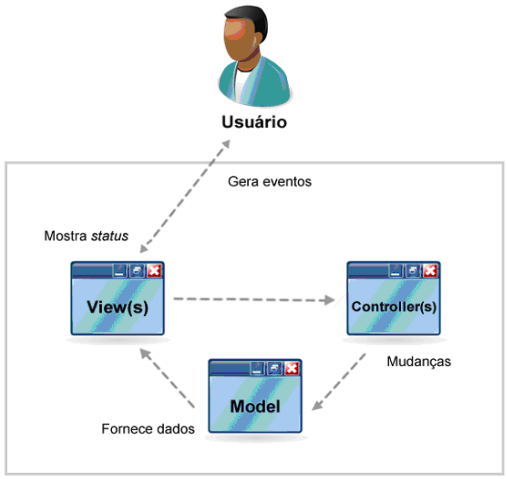
2.4.1. MVC

Model-View-Controler ou Modelo-Visão-Controlador, é um padrão de desenvolvimento que separa o sistema em camadas independentes com características e funções bem definidas. Dessa forma, cada camada pode ser alterada independentemente sem afetar as outras. Com isso é possível atingir os objetivos de eficiência, reutilização e facilidade de manutenção. As camadas do MVC são:

- **Modelo (Model):** contém os dados da aplicação e suas regras de negócio.
- **Visão (View):** contém a interface entre o sistema e o usuário.
- **Controlador(Controller):** contém o controle da interação entre modelo e visão.

Como demonstrado na Figura 2, a camada Modelo fornece os dados da aplicação para camada Visão, que por usa vez renderiza essa informação para o usuário. Já a camada Controlador, interpreta as ações do usuário e as mapeia para chamadas da camada Modelo.

Figura 2: Camadas do MVC



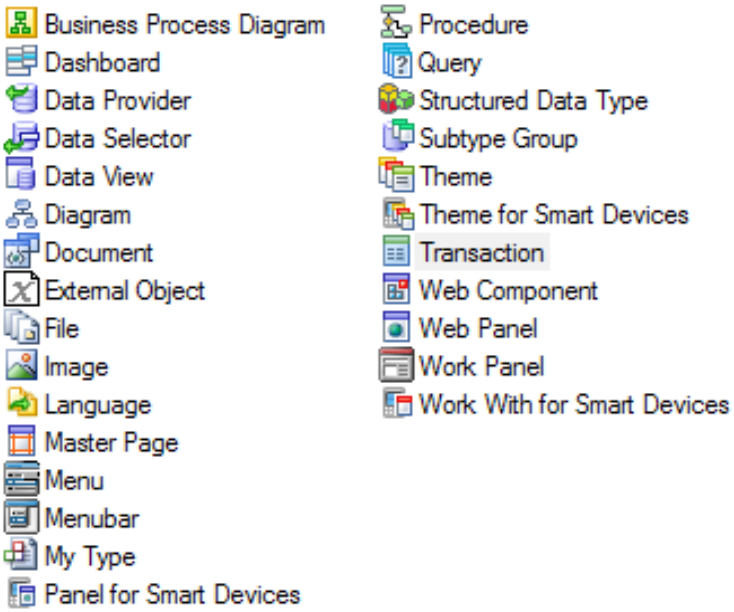
2.4.2. ADAPTANDO PADRÕES DE DESENVOLVIMENTO

Por se tratar de uma ferramenta CASE e possuir uma linguagem própria, desenvolver em GeneXus é algo singular. O código do GeneXus é estruturado em objetos, cada um com sua função. A figura 3 demonstra os tipos de objetos existentes.

Como os padrões de desenvolvimento são conceituais é possível adaptar parte de suas ideias para ser mais compatível com o projeto em desenvolvimento. No caso do Conex-e, algumas ideias do MVC foram aproveitadas:

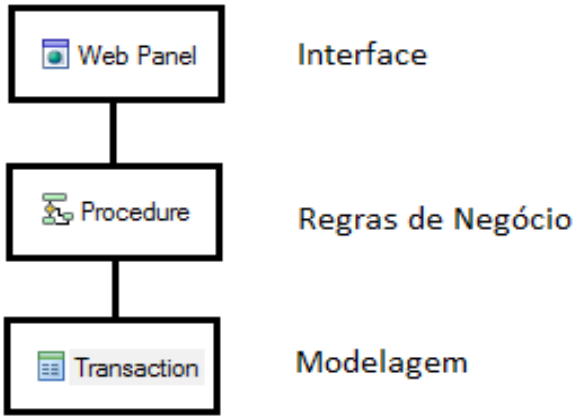
- Separar a interface, regras de negócio e modelagem.
- Dividir o sistema em camadas.

Figura 3: Tipos de objetos GeneXus



O GeneXus permite, através dos objetos do tipo Transação (Transaction), juntar interface, regras de negócio e modelagem no mesmo objeto. Dessa forma ele agiliza o processo de implementação. Porém essa prática não é indicada pois apenas uma pessoa pode alterar o objeto por vez e muito código comum entre objetos acaba sendo repetido diversas vezes. Para resolver esse problema é possível separar a interface, as regras de negócio e a modelagem em objetos diferentes.

Figura 4: Camadas do sistema

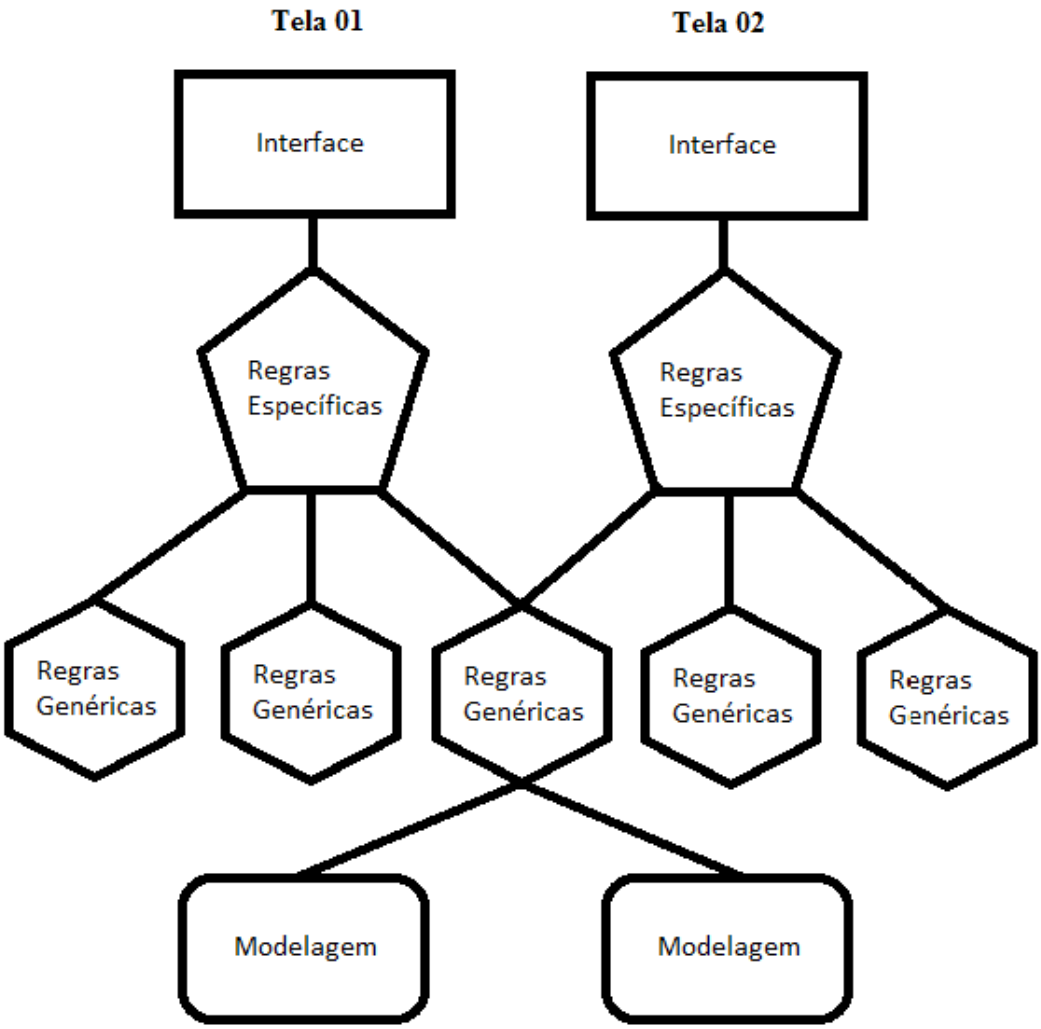


Como demonstra a figura 4, a interface do sistema fica em objetos do tipo Web Panel, que representam uma tela do sistema. Já as regras de negócio são implementadas em objetos do tipo Procedimento (Procedure), que são objetos específicos para regras de negócio e podem ser chamados facilmente por outros objetos. A modelagem das tabelas do banco de dados fica em objetos do tipo Transação, que armazenam os atributos utilizados no sistema e seus tipos.

Ao separar a interface, as regras de negócio e a modelagem torna-se possível mais de um desenvolvedor trabalhar na mesma tela ao mesmo tempo, um na interface, outro nas regras de negócio e outro na modelagem, por exemplo. Também é possível alterar uma camada sem interferir nas outras.

Para facilitar o reuso de código a camada responsável pelas regras de negócio foi dividida em mais camadas. Para tanto, cada tela do sistema possui, além dos objetos de interface e de modelagem, mais de um objeto de regras de negócio.

Figura 5: Camadas de regras de negócio



Como demonstra a Figura 5, cada tela possui uma camada de interface, duas ou mais camadas de regras de negócio e uma camada de modelagem. A divisão da camada de regras de negócio em camadas menores torna possível a reutilização de código. Para tanto, cada tela possui um objeto com regras específicas que se comunica com outros objetos para poder ter acesso as regras de negócio genéricas. Dessa forma as regras de negócio genéricas são modificadas apenas em um objeto, e não mais em todos os objetos em que elas são utilizadas. Além disso, torna-se possível que um desenvolvedor modifique as regras específicas de uma tela e outro modifique as regras genéricas do sistema ao mesmo tempo.

Exemplificando a nova estrutura a figura 6 demonstra a interface de um Web Panel simples. Já a figura 7 demonstra os eventos do Web Panel, onde há uma chamada para um procedimento com regras de negócio específicas, apresentado na figura 8 e um procedimento com regras de negócio genéricas, apresentado na figura 9. Por fim a figura 10 apresenta a Transaction da tela em questão, contendo a modelagem.

Figura 6: Interface do objeto WTRelatorioParametrosGerais

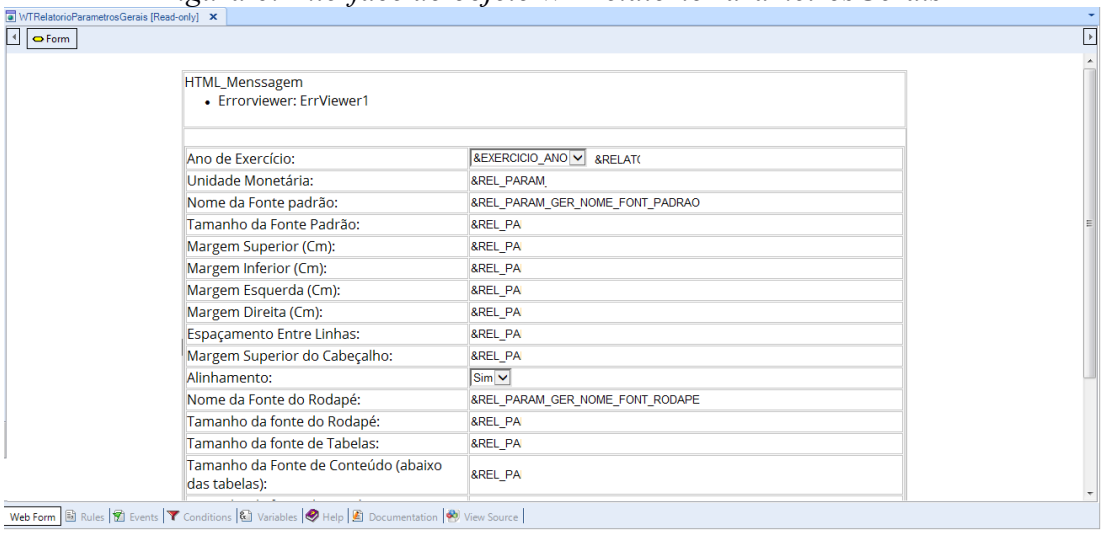


Figura 7: Eventos do objeto WTRelatorioParametrosGerais

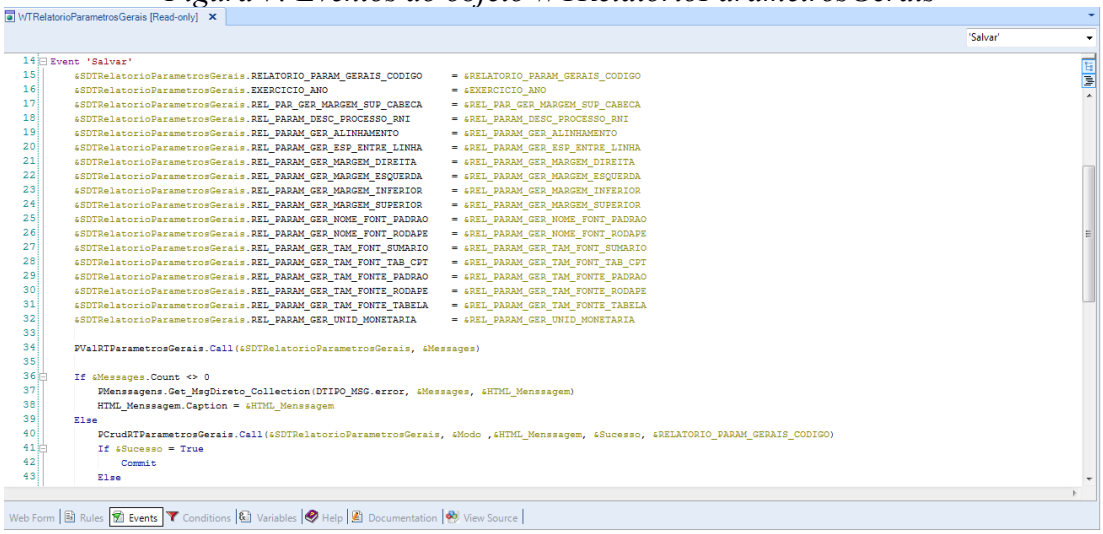


Figura 8: Procedimento específico PCrudRTParametrosGerais

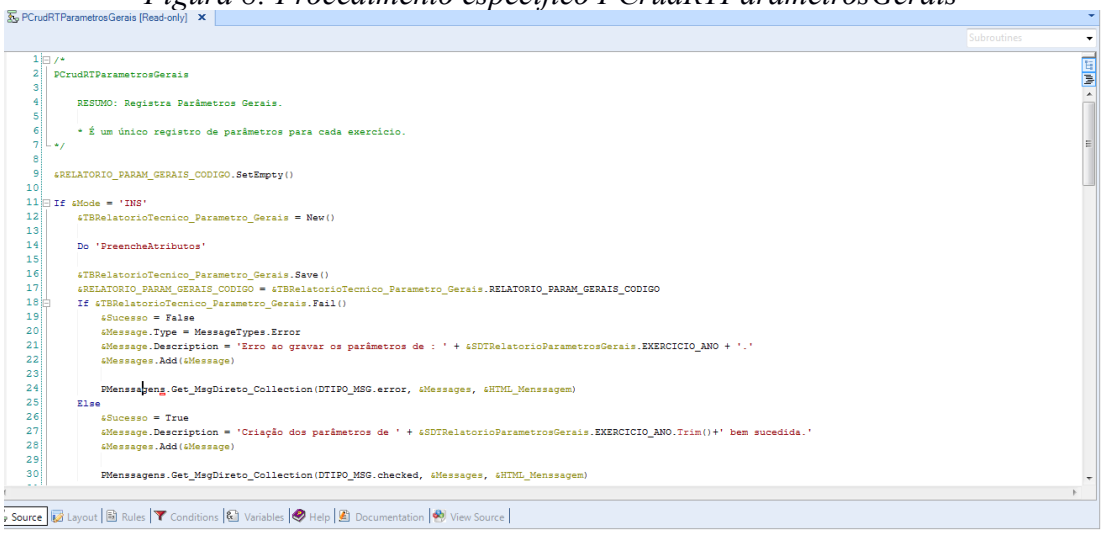
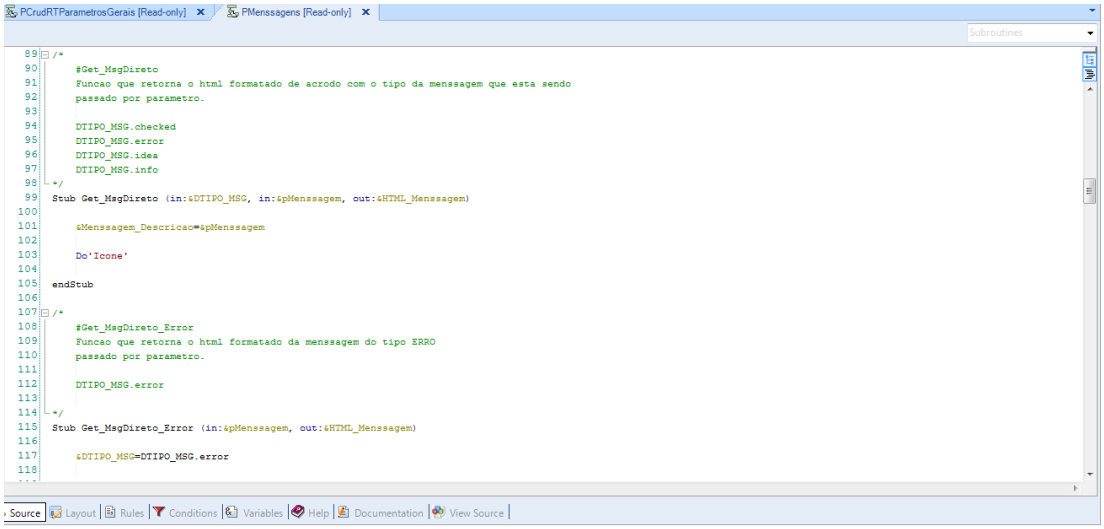


Figura 9: Procedimento genérico PMensagens



Com a adaptação do MVC para reestruturar o sistema, o número de objetos criados cresceu drasticamente. Para evitar a dificuldade de buscar os objetos tornou-se necessário a padronização dos nomes dos objetos. Para tanto, foi utilizado o conceito de códigos de barra, onde cada pedaço do código representa uma informação diferente. Por exemplo, um objeto contendo a interface de consulta de ordem de serviço se chamaria WPConsultaOS. Onde WP significa Web Panel, Consulta se refere ao tipo de tela e OS significa Ordem de Serviço. Da mesma forma a tela de cadastro de ordem de serviço se chamaria WPCadastroOS.

A divisão do sistema em camadas facilitou também os testes. A nova arquitetura permite que os testes sejam realizados somente nas partes em que o sistema foi modificado, diminuindo a necessidade de testes em todo o sistema. Dessa forma os testes se tornam mais rápidos e eficientes.

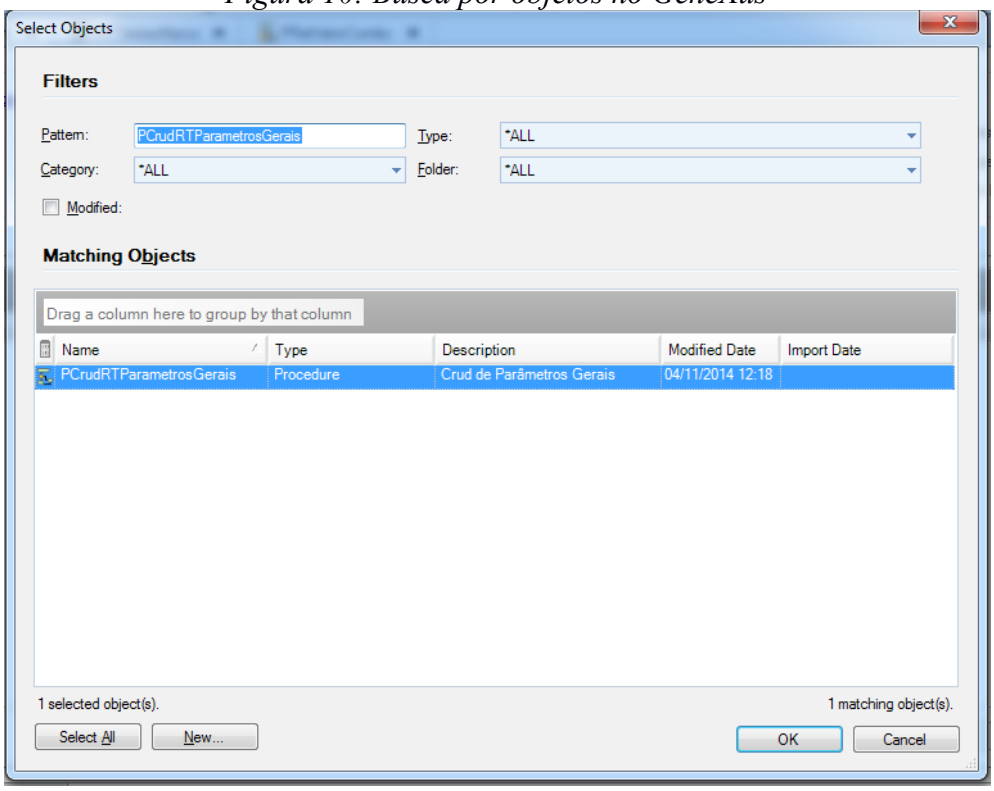
3. RESULTADOS

Com a utilização de padrões de desenvolvimento para reestruturar o sistema Conex-e foi possível atender todas as demandas existentes. As novas funcionalidades foram implementadas, em paralelo a reestruturação do sistema e já na nova arquitetura, sem muitas dificuldades.

Uma das novas funcionalidades implementadas consistia em uma sequência de processamentos em cima de uma ordem de serviço. Cada etapa desse processo dependia da anterior e só poderia prosseguir ao terminar a etapa atual. Com a nova estrutura do sistema as regras de negócio de cada etapa do processo foram implementadas em um objeto do tipo procedimento. Dessa forma, cada etapa pode sofrer manutenção independentemente.

A manutenção do sistema tornou-se mais simples, graças a nova arquitetura, sem complicar muito o processo de implementação. Conhecendo a estrutura dos objetos do sistema fica simples encontrar a fonte do problema. A produtividade foi mantida, com o apoio da ferramenta CASE GeneXus e o maior conhecimento do sistema pelos desenvolvedores.

Figura 10: Busca por objetos no GeneXus



Com a nova estrutura, caso fosse detectado um problema ao salvar as informações de parâmetros gerais do relatório técnico, por exemplo, em vez de ser necessário buscar o objeto da tela em questão e então buscar as regras de persistência em meio a todas as regras de negócio referentes a tela, o desenvolvedor precisa apenas abrir diretamente o objeto com as regras de negócio referentes aos parâmetros gerais do relatório técnico. Com a novo padrão de nomenclatura de objetos essa tarefa se tornou ainda mais simples bastando buscar pelo objeto *PCrudRTParametrosGerais*. Nesse caso, *P* se refere ao objeto do tipo Procedimento, *Crud* ao tipo de regras de negócio de persistência e *RT* a relatório técnico. A Figura 10 demonstra a busca por esse objeto.

Os testes se tornaram mais simples, rápidos e eficientes com a nova estrutura de objetos. Graças a divisão do sistema em camadas, os testes são mais pontuais diminuindo a necessidade de testes de todo o sistema. Além disso houve a implementação de testes automatizados dentro do próprio sistema, exemplificados nas figuras 11 e 12.

Figura 11: Interface de uma rotina de teste automatizado

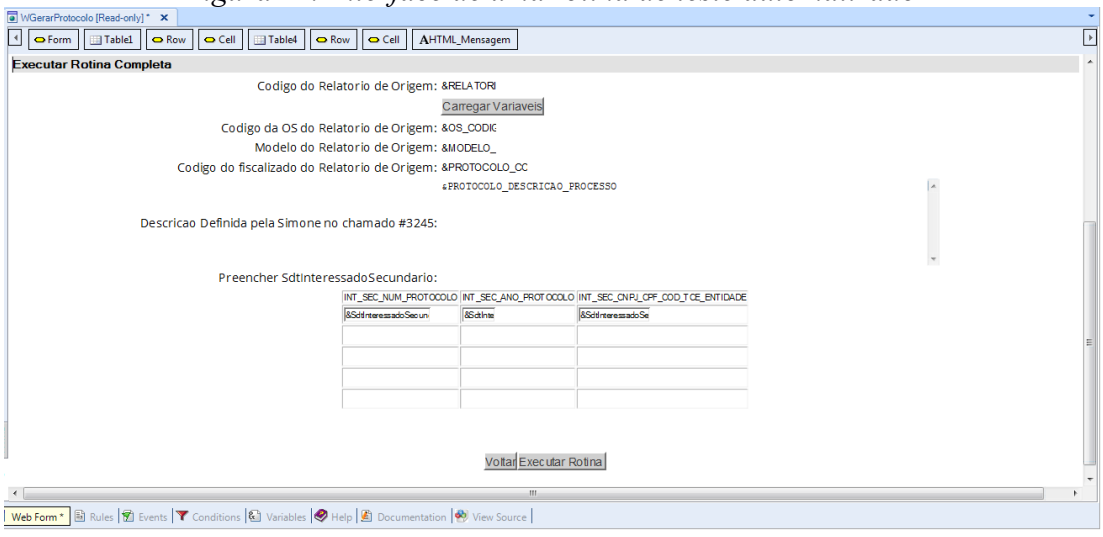
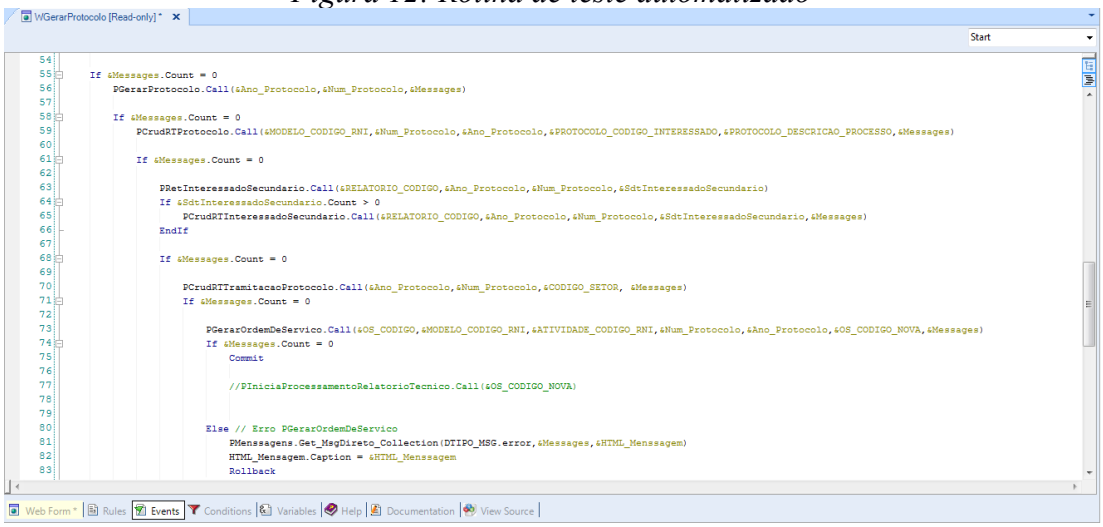


Figura 12: Rotina de teste automatizado



Com a divisão de regras de negócio em camadas, objetos podem sofrer manutenção independentemente. Dessa forma, seguindo o exemplo anterior, ao se alterar as regras de persistência dos parâmetros gerais do relatório técnico se torna necessário testar apenas essa funcionalidade, pois suas regras estão separadas das outras presentes na tela em questão. Anteriormente era necessário testar todas as funcionalidades da tela para conferir se houve alguma alteração indesejada durante a manutenção, pois todas as regras de negócio e interface estavam presentes no mesmo objeto.

Além disso, a adesão de novos integrantes a equipe se tornou menos complicada pois a nova arquitetura facilitou o entendimento do sistema como um todo. Com a adoção da padronização de nomes de objetos a busca por código ficou bem mais simples, pois cada objeto possui sua função específica e seu nome representa isso.

4. DIFICULDADES ENCONTRADAS

Pelo fato do sistema abordado neste relatório ser baseado em duas tecnologias, em vários momentos durante a reestruturação houve dúvidas sobre qual tecnologia deveria implementar determinadas funcionalidades. Ambas tecnologias possuem suas vantagens e desvantagens, além disso, a demanda por novas funcionalidades e a arquitetura existente dificultavam ainda mais a tomada de decisões. Muitas reuniões foram necessárias e propostas foram apresentadas até a definição da arquitetura final, apoiada pelos novos padrões de desenvolvimento.

Uma das principais deficiências do sistema eram os testes que em sua maioria eram totalmente manuais e demorados. Isso criou a necessidade de uma forma automatizada e ágil de se efetuar testes. Durante as reuniões duas ferramentas de teste unitário foram citadas, uma para cada tecnologia. No lado Java o conhecido e amplamente utilizado JUnity, já no GeneXus o nem tão conhecido e utilizado GXtest. No entanto, nenhuma solução pode atender de forma eficaz o que era proposto. Para poder utilizar o JUnity, a maior parte do sistema precisaria ser portada do GeneXus para o Java, algo que se mostrou inviável no tempo disponível. Já no caso do GXtest, ele se mostrou uma ferramenta com suporte e documentação insuficientes, além de ser contraindicado por muitos que o utilizaram. Para poder atender a demanda de testes a solução encontrada foi criar rotinas de teste dentro do próprio sistema através do GeneXus, para agilizar e automatizar o que fosse possível.

Além dos testes unitários, houve dificuldade também com a função de debug do GeneXus, pois ela não é conhecida nem utilizada amplamente. Isso se dá por um outro problema enfrentado com a ferramenta GeneXus, que é a dificuldade de se encontrar publicações sobre seus componentes e funcionalidades. A documentação muitas vezes se mostra insuficiente para dar suporte ao desenvolvimento, tornando assim o uso de muitas funcionalidades, como o debug, inviável. Para contornar o problema com o debug da ferramenta uma prática já utilizada anteriormente foi mantida, os logs de sistema. Logs são gerados em meio ao código para rastreio das variáveis e assim acompanhar a execução do código que está sendo desenvolvido.

5. CONCLUSÕES

A utilização de padrões de desenvolvimento se mostrou uma prática muito efetiva. Por se tratar de conceitos e boas práticas de desenvolvimento amplamente testadas há a garantia que sua aplicação será vantajosa. Porém, é sempre necessário avaliar o custo de se aplicar um determinado padrão de desenvolvimento para saber se as vantagens são maiores que as desvantagens para o sistema em questão.

Os padrões são conceituais e podem ser adaptados e unidos para atender as mais diversas demandas existentes. Portanto, é interessante para o desenvolvedor conhecer os padrões de desenvolvimento, mesmo que superficialmente, para saber como resolver de forma eficaz os problemas e dificuldades encontradas durante a implementação de um sistema.

No sistema Conex-e o uso de um padrão de desenvolvimento adaptado supriu todas as demandas existentes. As novas funcionalidades foram implementadas em uma arquitetura melhor que a anterior. A manutenção e o teste do sistema se tornaram mais simples, eficazes e menos custosos.

Com a realização deste relatório foi possível perceber que existem muitos padrões de desenvolvimento que facilitam o trabalho desenvolvedores e tornam os sistemas mais eficientes. Nem todos os padrões puderam ser pesquisados e abordados durante a realização desse relatório. Dessa forma, torna-se desejável futuras pesquisas sobre todos os padrões de desenvolvimento existentes.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, Cristopher; ISHIKAWA, Sara; SILVERSTEIN, Murray. 1977. *A Pattern Language – Towns, Buildings, Construction.* Oxford, Inglaterra : Oxford University Press.

AQUINO, Bruno. GeneXus – O que é e para que serve. Disponível por http://www.itfriends.org/genexus-o-que-e-e-para-que-serve/

SHVETS, Alexander. Design Patterns. Disponível por www em http://sourcemaking.com/design_patterns

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* EUA : Pearson Education.

FARIAS, Adalberto; 2001. *Ferramentas CASE: Suporte, Adoção e Integração*. Recife – PE. Monografia de Engenharia de Software - Universidade Federal de Pernambuco, Centro de Informática.

KURBEL, Karl; SCHNIEDER, Thomas. 1994. *Integration Issues of Information Engineering Based I-CASE Tools*. In: Fourth International Conference on Information Systems Development, Bled, Slovenia, September 20-22, 1994. University of Münster, Institute of Business Informatics, Alemanha.

MANLEY, Gary Wayne; 1990. *The Classification and Evaluation of Computer-Aided Software Engineering Tools*. Monterey, California. Tese de mestrado. Naval Postgraduate School, National Science Foundation.