Desenvolvimento de uma metodologia de Gerenciamento de Configuração de Software integrada ao Visual Studio

Ramon Fernandes Mendes

Universidade de Caxias do Sul
Centro de Computação e Tecnologia da Informação
ramon@midi.rs

1. Introdução

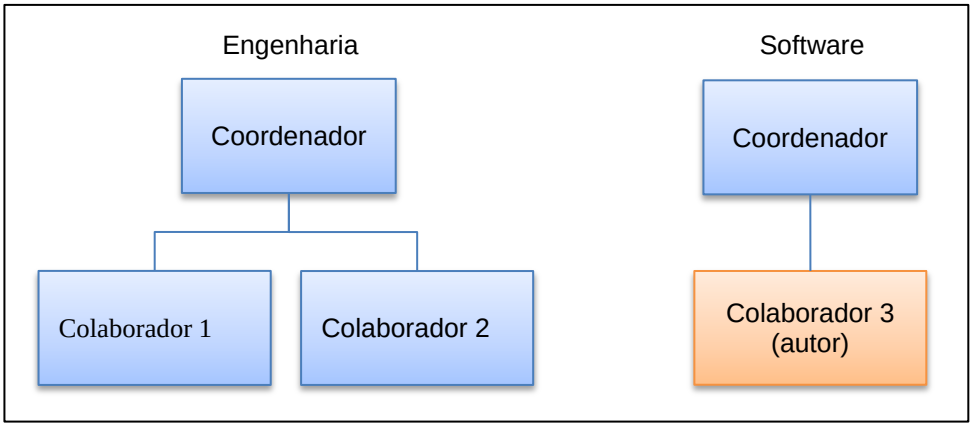
Nas empresas de software, a busca pelo aumento da produtividade e pela melhoria contínua da qualidade tem levado a utilização de ferramentas de apoio ao processo de desenvolvimento como, por exemplo, ferramentas de colaboração entre membros do projeto, controle de versão, testes automatizados, integração contínua, controle de *tickets*. Modelos de qualidade software, como MPS.BR (SOFTEX, 2009) e CMMI (SEI, 2001), definem a área de processo de Gerenciamento de Configuração de Software (GCS) que determina o controle de artefatos do projeto, o que inerentemente está ligado a utilização de tais ferramentas.

A Akron é uma micro-empresa que atua na área de automação industrial com foco no rastreamento de produtos, sensoriamento da linha de produção e gestão da produção. O negócio da empresa é baseado em prestação de serviço no desenvolvimento customizado de soluções para o cliente. Dentre as tecnologias adotadas na empresa se integra a tecnologia RFID que permite identificar produtos à distância através de etiquetas eletrônicas (*tags*). Está tecnologia é diferencial nas soluções da empresa sendo normalmente integrada aos sistemas de informação do negócio, exigindo um grande esforço de análise e desenvolvimento de *software* e também de implantação do hardware específico (antenas, leitores, coletores).

A organização funcional da empresa é enxuta, integra 2 coordenadores (sócios da empresa) e 3 colaboradores. Existem 2 áreas funcionais: engenharia e *software*. A engenharia realiza atividades de desenvolvimento de *hardware* e *firmware*, projeto, aquisição, instalação e manutenção de equipamentos, possui 1 coordenador e 2 engenheiros subordinados. A área de *software* desempenha todas

as atividades de desenvolvimento, configuração e instalação de *software*, possui 1 coordenador, e 1 desenvolvedor, que é a posição que o autor deste trabalho desempenha. Na Figura 1 é apresentada esta estrutura.

Figura 1 – Estrutura funcional da Akron



Fonte: autor

A área de *software* é a principal responsável por desenvolver, testar e entregar novas versões de *software* e, embora a engenharia desempenhe esses papéis em menor parte, cabe a área de *software* propor melhorias para o aprimoramento contínuo no desempenho de tais funções. Portanto, no desenvolvimento deste estágio, o foco de atuação será nesta área, especificamente nas funções de desenvolvimento, assumindo-se este papel de implantar melhorias ao definir e modelar um processo de desenvolvimento, atuando no âmbito de toda empresa buscando envolver todos os colaboradores.

2. Levantamento da Situação Atual

Na empresa se percebe duas situações diferentes de trabalho na área de *software*: desenvolvimento de projeto e desenvolvimento de produto. Projetos são sistemas desenhados especificamente segundo a demanda do cliente e possuem bem definido um escopo, e se identifica claramente o inicio e o fim da etapa de desenvolvimento. Já o produto é um sistema que a empresa oferece pronto para uso, e que está sempre sob constante aprimoramento, normalmente não tendo uma previsão para o fim do desenvolvimento.

Cada projeto, por se tratar de um sistema personalizado para um único cliente, possui toda uma primeira etapa de análise, desenvolvimento e testes, e só posteriormente é entregue e colocado em produção. Já o produto possui diversos clientes para um mesmo sistema, tendo versões do sistema em produção, e uma linha de desenvolvimento baseada nas demandas dos atuais e futuros clientes. Para a implementação, a principal ferramenta utilizada na empresa é o Visual Studio juntamente com as tecnologias C#, .Net, framework MVC5 e banco de dados SQLServer.

O desenvolvimento dos sistemas não possui um processo bem definido. Existe basicamente um controle interno das funcionalidades pendentes ou já implementadas, feito através de uma planilha Excel compartilhada entre a equipe. Este mecanismo serve basicamente para acompanhar as tarefas que ainda necessitam ser executadas. Ocorre que, como este sistema é muito rudimentar, acaba raramente sendo atualizado, embora seja o único controle existente quanto o andamento do desenvolvimento. Ocorre situações em que determinada funcionalidade é implementada mas não marcada na planilha e com isso, posteriormente, não se sabe se já foi concluída, quem concluiu, se já foi implantada no cliente, etc. Também não existe um controle de versão de código que permita rastrear quais partes do sistema foram alteradas para cumprir a funcionalidade.

Atualmente um dos principais produtos da empresa é um sistema de gerenciamento de frotas que, além de fornecer rastreamento veicular por GPS, apresenta um sistema de software para gerenciamento dos veículos rastreados através de diversos módulos como: manutenção, financeiro, agendamento de visitas, checklist dos motoristas, controle de pontos de interesse e relatórios. Esse produto já possui 2 anos de desenvolvimento tendo como objetivo ser uma pacote fechado para venda ao cliente, o qual pode escolher os módulos prontos que deseja adquirir, embora possa também requisitar funcionalidades customizadas. O sistema é disponibilizado on-line para o cliente acessar através da internet e já sair utilizando. Existe um único servidor que hospeda uma aplicação Web separada para cada cliente. Cada aplicação possui um banco de dados exclusivo.

Este produto possui uma única base de código fonte, que é modificada de forma concorrente pelos 2 desenvolvedores. A junção dos fontes é realizada de

forma manual, arquivo por arquivo. Essa etapa eventualmente apresenta problemas de as modificações não serem corretamente combinadas necessitando posterior revisão e retrabalho. Atualmente essa mesma base de código possui funcionalidades específicas para cada cliente, ou seja, este único produto é capaz de suportar as customizações presentes para cada cliente. Ao colocar em produção, as funcionalidades são habilitadas/desabilitadas conforme o cliente. Atualmente esta solução é adequada devido ao número reduzido de clientes, mas percebe-se que ela não é escalável, necessitando talvez manterem-se versões separadas por cliente já que as necessidades de novas atualizações são constantes.

3. Objetivos e Proposta de Solução

O objetivo do trabalho é iniciar a Gerência de Configuração na empresa, implantando uma infraestrutura base para controle de versão e controle de modificações, e definindo e institucionalizando um processo de *software*.

A principal motivação é o fato de que o principal produto da empresa está em plena ascensão com previsão de que eventualmente terá de suportar dezenas de diferentes clientes, muitos com versões personalizadas do produto. Como o produto está ainda em uma fase inicial de desenvolvimento este é um momento oportuno para definir um processo adequado, que controle as constantes atualizações na base de código através do versionamento, controle quais mudanças são realizadas nos itens de configuração, e que defina como são realizadas as entregas das novas versões do *software.*

Como a empresa não adota nenhuma metodologia de desenvolvimento, os benefícios esperados para a empresa são de que a GCO seja um meio prático de buscar um melhor processo de *software*, melhorando a qualidade do produto e garantindo a satisfação do cliente.

Segundo Softex (2012) o sistema de controle de modificações “tem a função de executar a função de controle da configuração de forma sistemática, armazenando todas as informações geradas durante o andamento das solicitações de modificação e relatando essas informações aos envolvidos”.

Já o sistema de controle de versão “permite que os itens de configuração sejam identificados, segundo estabelecido pela função de identificação da configuração e que eles evoluam de forma distribuída e concorrente, porém disciplinada”.

A ideia principal do estágio é primeiramente adotar uma ferramenta de controle de versão. A partir deste controle, se conseguirá definir *baselines* que “oferecem uma ‘memória’ da situação dos produtos de desenvolvimento de software” (SOFTEX, 2012), e quais os seus itens de configuração. Finalmente o processo de GCS obtido com a utilização dessas ferramentas poderá ser desenhado com a notação BPMN.

A proposta de solução contempla as seguintes atividades, segundo o cronograma apresentado na Figura 2:

1. Estabelecer os objetivos da GCO na empresa: requisitos, itens de configurações, e ferramentas que serão implantadas e avaliadas;

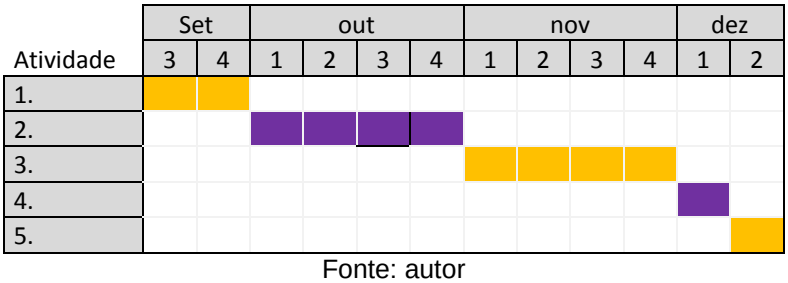
2. Escolher e adotar uma ferramenta para controle de versão;

3. Escolher e adotar uma ferramenta de controle de modificações;

4. Desenhar e descrever o novo processo e os resultados obtidos;

5. Redigir trabalho final.

Figura 2 – Cronograma do estágio



Fonte: autor

4. Desenvolvimento da proposta

A primeira atividade do estágio, iniciada no dia 20/09/2014, envolveu uma discussão com a área de software sobre a proposta do estágio e um momento de definição sobre os objetivos que seriam alcançados.

Conforme se esclareceu, o principal objetivo do estágio será adoção de ferramentas para a atividade de controle de configuração da GCS, de modo a desenvolver padrões e procedimentos para gerenciar a evolução especificamente do produto de Gestão de Frotas.

Conforme se identificou na literatura, dentro da GCS, é a atividade de controle de configuração que se encarrega de acompanhar a evolução dos itens de configuração e definir as *baselines*, que são conjuntos de itens de configuração formalmente aprovados, agrupados em determinados pontos do ciclo de vida de desenvolvimento e manutenção, para atender propósitos específicos e que servem de base para etapas posteriores do projeto (YOSHIDOME, 2009).

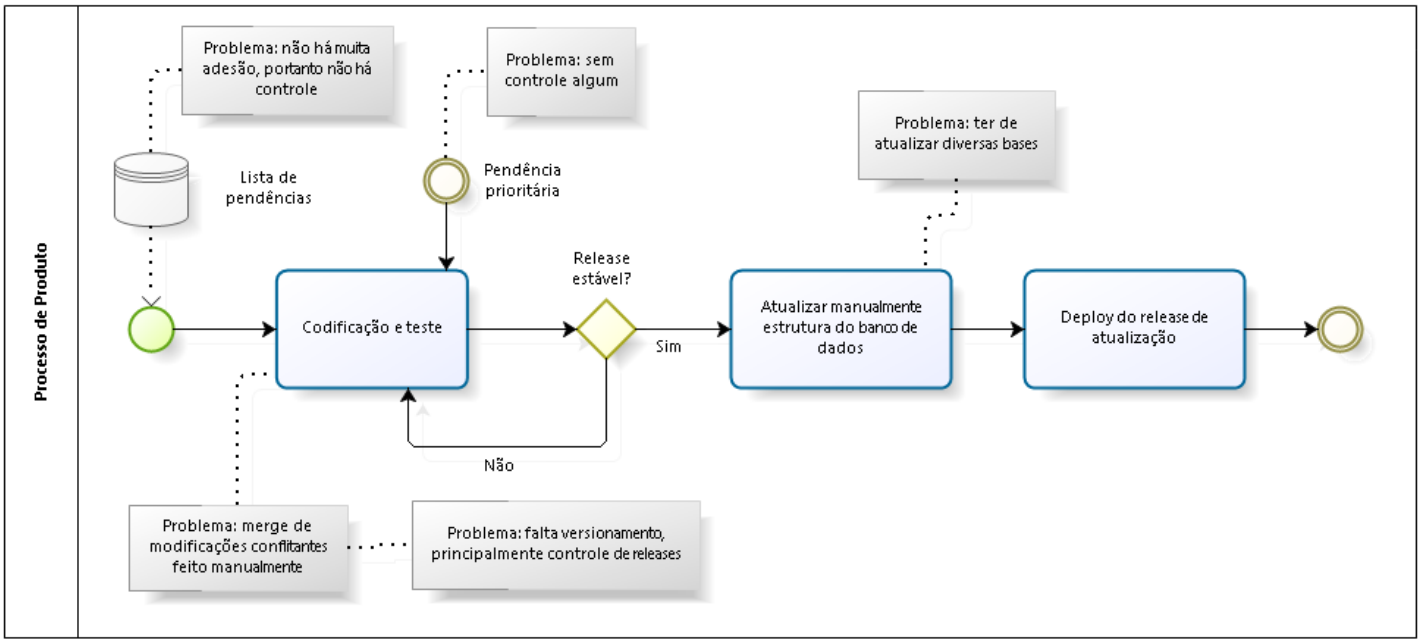
Para Yoshidome, no controle de configuração, pode-se identificar duas tarefas: o controle de versão e o controle de modificações. O controle de versão objetiva identificar e acompanhar o desenvolvimento de diferentes versões ou releases de um sistema. O controle de modificação estabelece atividades para conduzir uma modificação desde sua solicitação até a verificação de sua implementação e mantém os interessados a par de quaisquer modificações em itens de configuração.

Na empresa, como itens de configuração, definiu-se que estes contemplariam os produtos de desenvolvimento de software, ou seja, código fonte e arquivos relacionados que compõe o produto, e que não são entregues ao cliente, mas sim usados na geração do software e na documentação interna. Já as *baselines* serão as versões de release do software, as quais são aprovadas pela equipe e colocadas em produção.

Como o atual foco da empresa é no desenvolvimento do produto de gestão de frotas, se mapeou o processo atual desempenhado na sua construção e os

pontos onde ocorrem problemas, conforme pode ser visualizado na Figura 3. Nas atividades do estágio serão discutidos estes problemas e as soluções encontradas.

Figura 3 – Processo atual de desenvolvimento de produto



Fonte: autor

4.1 Requisitos

Com base na discussão realizada da empresa foram definidos os requisitos para o sistema:

Requisitos para o sistema de controle de versão:

R1 – Controlar os itens de configuração, permitir buscar/visualizar um histórico de alterações, rastrear as alterações, criar ramificações, controlar diferenças e a resolução de conflitos;

R2 – Permitir registrar as *baselines* (milestones, versões de release ou tags);

R3 – O acesso ao repositório deve ser privado aos usuários autorizados;

R4 – A ferramenta deve possuir integração ao Visual Studio e fornecer de modo intuitivo a execução das atividades básicas de controle de versão: commit, merge, clone, pull, push, fetch;

R5 – Possuir uma interface Web, que suporte vários projetos e permite acompanhar cada projeto, fornecendo estatísticas sobre o controle de versão;

R6 – Possuir integração com o sistema de controle de modificações;

R7 – Controlar a evolução da estrutura do banco de dados e o deploy nas bases em produção dos clientes;

Requisitos para o sistema de controle de modificações:
R8 – Possuir uma interface Web para gerenciamento das tarefas com acesso privado;
R9 – Possuir integração com o sistema de controle de versão;
R10 – Permitir registrar tarefas específicas por pessoa e permitir categorizar o tipo de tarefa (por projeto ou cliente) e sua prioridade;
R11 – Permitir rastrear modificações que resultaram no atendimento de uma tarefa e liberação/release de uma nova versão do sistema;

Não foi definido como requisito uma restrição quanto ao sistema ser off-line, instalado localmente na empresa, ou on-line, como serviço na nuvem, sendo esta uma das principais escolhas a ser definida. Também não é requisito que o sistema seja gratuito. Ambos os sistemas devem estar integrados, o que na prática significa que o sistema de controle de modificações deve suportar o SCV escolhido.

4.2 Sistema de controle de versão

A Atividade 2 do estágio contemplou escolher, adotar e avaliar um conjunto de ferramentas para controle de versão. Esta seção aborda a escolha do SCV, o Git, a adoção através de um repositório local, e a utilização do Visual Studio para execução das tarefas do fluxo de trabalho. Também aborda o controle de versão do banco de dados através das migrações.

4.2.1 Seleção do SCV

Um sistema de controle de versão (SCV) pode ser classificado em centralizado ou distribuído. No modelo centralizado existe apenas um repositório central e várias cópias de trabalho. No modelo distribuído, existem vários repositórios autônomos e independentes, um para cada desenvolvedor, e cada um desses repositórios possui uma área de trabalho acoplada a ele. SCVs distribuídos são mais modernos pois surgiram depois dos SCVs centralizados e os desenvolvedores puderam corrigir erros já conhecidos e desenvolver novas

funcionalidades que os próprios usuários já relatavam como essenciais (FREITAS, 2010).

Dos sistemas de controle de versão que trabalham de forma distribuída, dois que mais ganharam adeptos e tiveram maior aceitação foram o Mercurial, criado por Matt Mackall e o Git, criado por Linus Torvalds, o também criador do sistema operacional Linux (FREITAS, 2010).

Para Dorsey (2014), Git e Mercurial são opções praticamente equivalentes de SCV. Na maioria dos casos, a escolha pode ser feita baseada na preferência, devendo-se experimentar cada um e verificando-se qual causa menos fricção no fluxo de trabalho diário. No entanto Git é uma ferramenta que possui vantagens em termos de ferramentas de suporte como clientes *desktop*, ferramentas de integração e serviços disponíveis na nuvem.

Para seleção do SCV a ser utilizado, o principal critério adotado foi a conveniência de nativamente já haver integrado no Visual Studio o suporte para o SCV em questão, que é o Git. Como o uso de um SCV é algo novo para os integrantes da empresa, uma ferramenta ágil, e que naturalmente pudesse ser integrada no processo de desenvolvimento se mostrou indispensável para garantir a adesão ao controle de versão.

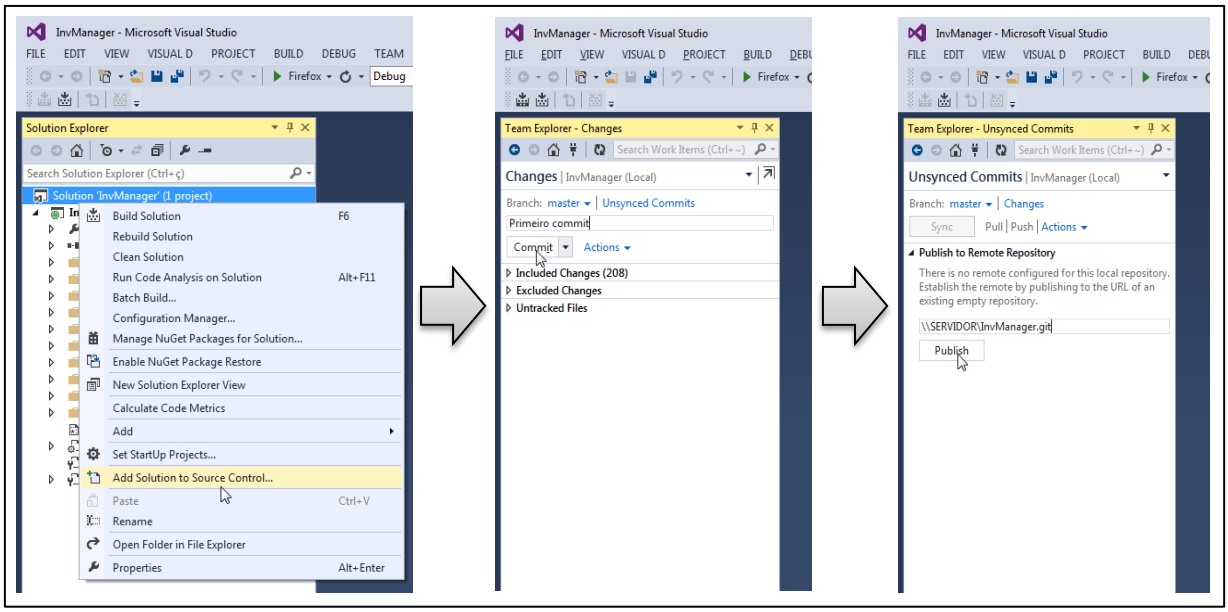
Segundo OLAUSSON (2013), a partir do Visual Studio 2012, a Microsoft está comprometida em manter o suporte ao Git oferecendo um serviço integrado ao Visual Studio, suportando repositórios Git do TFS ou repositórios Git de terceiros. O Visual Studio 2013, a versão usada na empresa, possui um excelente suporte ao Git, com todas as opções usuais de versionamento (commit, branching, histórico, merging) disponíveis de forma fácil e intuitiva.

4.2.2 Adoção do controle de versão

Na semana do dia 01/10/2014 iniciou-se a adoção do Git. Usou-se o Git instalado em um servidor da própria empresa para aprender os conceitos básicos de manutenção do repositório. Criou-se um repositório central Git (comando `git init –bare`) em um diretório compartilhado localizado num servidor com Windows 7. Como demonstrado na Figura 4, através do Visual Studio colocou-se a *solution* (projeto) e todo o código fonte sob versionamento Git (1), adicionou-se todos os

arquivos ao índice (*included changes*) e criou-se a primeira revisão através da opção de *commit* (2), e informando-se o endereço do repositório central, através do comando publish pode-se realizar o *push* da primeira revisão ao repositório central (3) que é simplesmente uma pasta compartilhada do Windows.

Figura 4 – Processo de versionamento da solução no VS

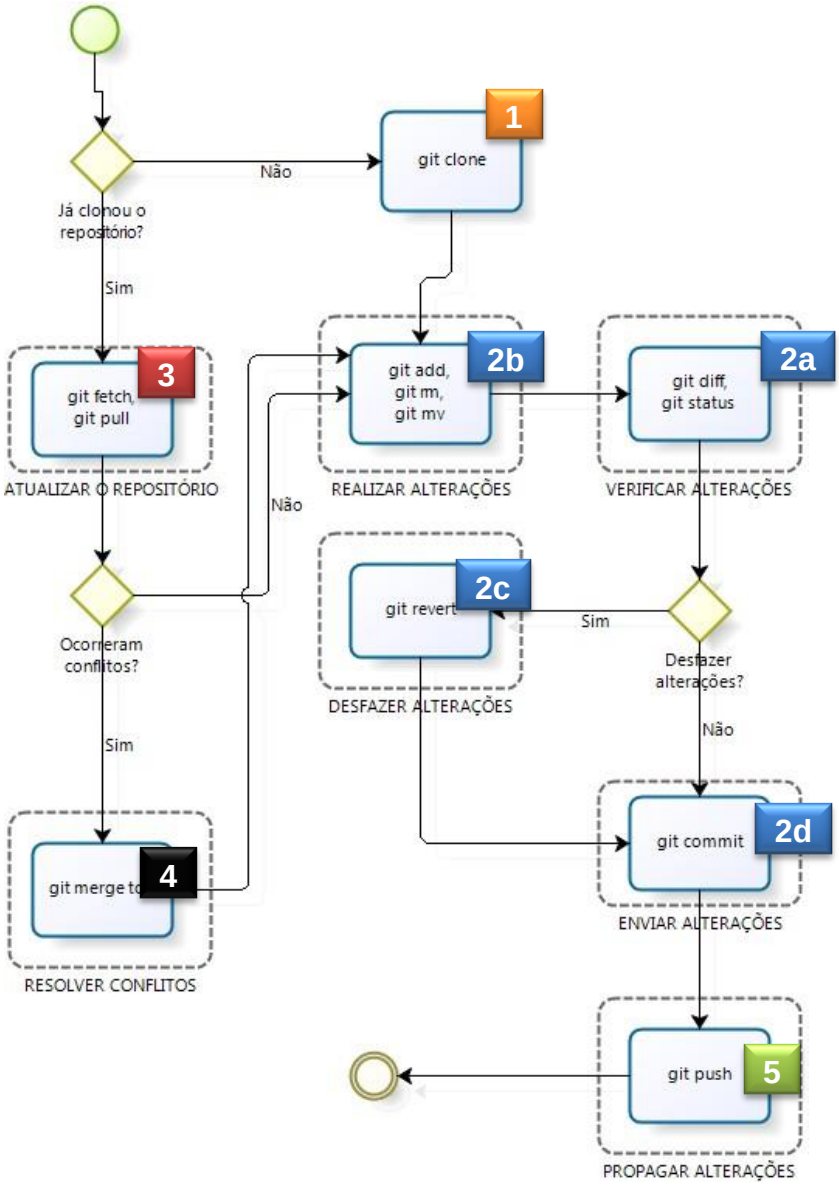


Fonte: autor

Neste trabalho decidiu-se usar a topologia cliente-servidor do Git, utilizando um servidor central, pois essa topologia é requisito básico para a ferramenta de controle de modificação. Além disso, o plugin Git do Visual Studio 2013 exige que se siga essa topologia para sincronismo das áreas de trabalho (*push* e *pull*), algo que foi descoberta durante a utilização da ferramenta.

Para entender o funcionamento do Git, pesquisou-se os comandos e o ciclo básico de trabalho desempenhado por um desenvolvedor no seu dia a dia e, a partir disso se mapeou como essas operações seriam executadas na empresa através do Visual Studio. A Figura 5 apresenta o fluxo do Git e uma numeração identificando cada etapa.

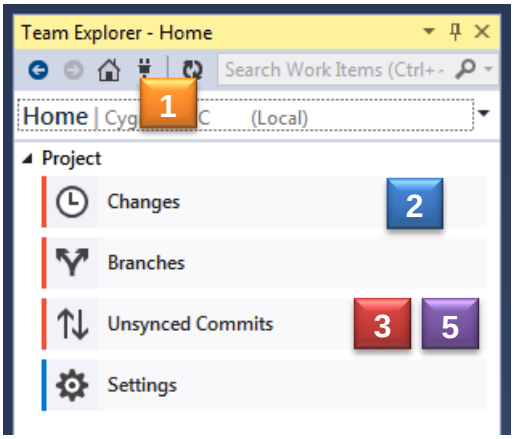
Figura 5 – Fluxo de trabalho do Git



Fonte: adaptado de FREITAS (2010)

No Visual Studio foi possível desempenhar esse fluxo através do painel Team Explorer (Figura 6) que fornece de forma intuitiva estas opções que, de outro modo, necessitariam ser executadas pela linha de comando. As figuras e descrições a seguir apresentam o uso da ferramenta conforme se deu no dia-a-dia da empresa.

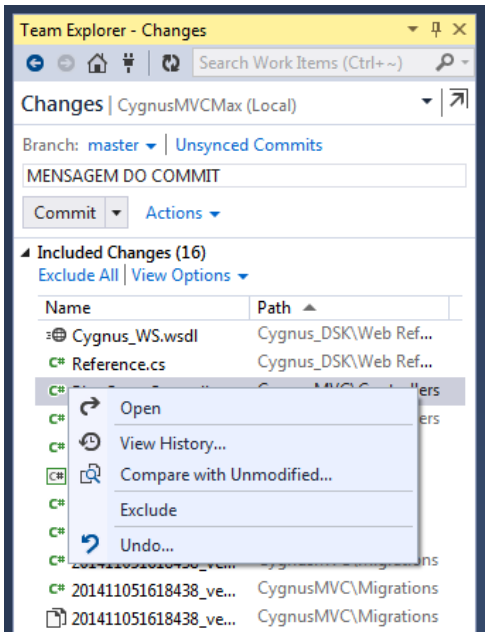
Figura 6 – Painel Team Explorer



Fonte: autor

A partir do botão ‘Connect’, conecta-se a um repositório Git já existente no computador, ou caso ainda não exista, há a opção de executar a operação de clonar (`git clone`) um novo repositório (etapa 1 do fluxo). Através do link 2, abre-se o sub-painel ‘Changes’ (Figura 7) que exibi quais os arquivos foram modificados (etapa 2a e 2b do fluxo) pelo desenvolvedor, podendo-se desfazer a alteração (etapa 2c) individualmente por arquivo, e por fim criar um novo *commit* com essas alterações (etapa 2d), sendo necessário digitar uma mensagem que facilmente identifica a nova revisão sendo salva.

Figura 7 – Painel Changes

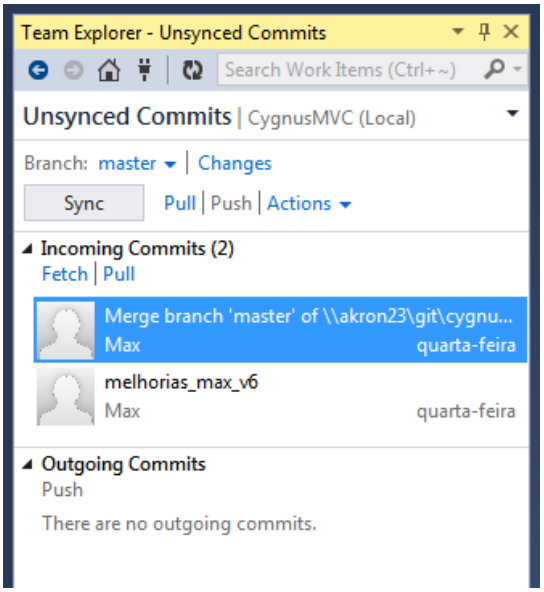


Fonte: autor

No sub-painel ‘Unsynced Commits’ (Figura 8) estão todas as opções de sincronizar as revisões com o servidor. No Git, sempre antes de poder enviar (*push*) os *commits* para o servidor, é necessário assegurar que o repositório local esteja atualizado para a última revisão do repositório central através da operação de *pull*.

Pode-se primeiramente executar a operação *fetch* para baixar os *commits* remotos que devem ser adicionados ao repositório local. Na Figura 8 observa-se que 2 novos *commits* que estão pendentes.

Figura 8 – Painel Unsynced Commits



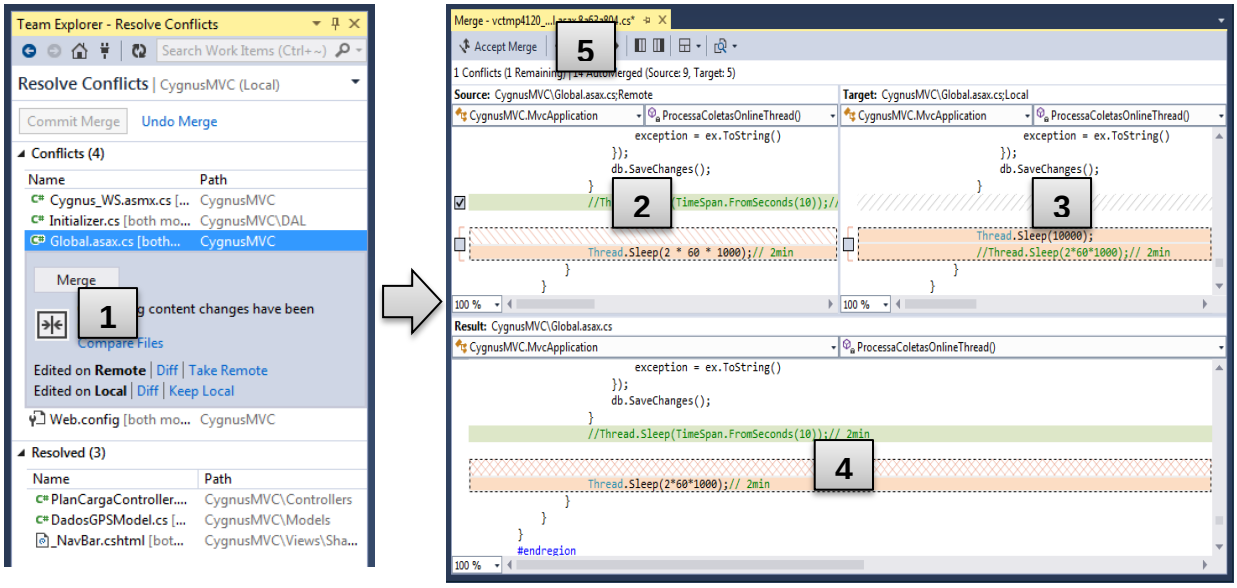
Fonte: autor

A opção *pull* realiza o merge desses *commits* ao repositório local (etapa 3 do fluxo). Ao fim desse comando, caso não houver conflitos, o repositório local estará sincronizado com a revisão do servidor.

Na empresa é comum, no entanto, ocorrer a situação de conflito entre arquivos do código-fonte editados simultaneamente pelos 2 desenvolvedores. Isso demanda do programador ter de resolver o conflito através do merge de ambos arquivos e a criação de um novo *commit*. O Visual Studio oferece um excelente mecanismo para esta tarefa, conforme mostrado na Figura 9. No painel são listados todos os arquivos conflitantes. Clicando no botão Merge (1), é aberta uma nova janela onde ambos os arquivos são mostrados lado a lado (2 e 3). Cada trecho de código conflitante possui uma checkbox de ambos os lados, o que permite ao programador escolher qual lado possui o conteúdo correto. A parte 4 exibi em tempo

real o novo conteúdo do arquivo que será obtido com o *merge*, que é finalizado através do botão 5.

Figura 9 – Resolução de conflitos através da ferramenta de merge



Fonte: autor

4.2.3 Controle de versão do banco de dados através de migrações

Outro mecanismo de controle de versão que foi adotado na empresa através do estágio foi o controle de migrações do banco de dados, de modo a atender o requisito R7. Um problema enfrentado no desenvolvimento do produto de rastreamento de veículos se deve a característica deste possuir versões diferentes em produção para cada cliente, sendo que cada um possui uma instalação única e exclusiva. Normalmente novas atualizações são feitas sob demanda de forma separada para cada instalação da aplicação, ou seja, novas versões do produto nunca são entregues de forma simultânea entre todos clientes.

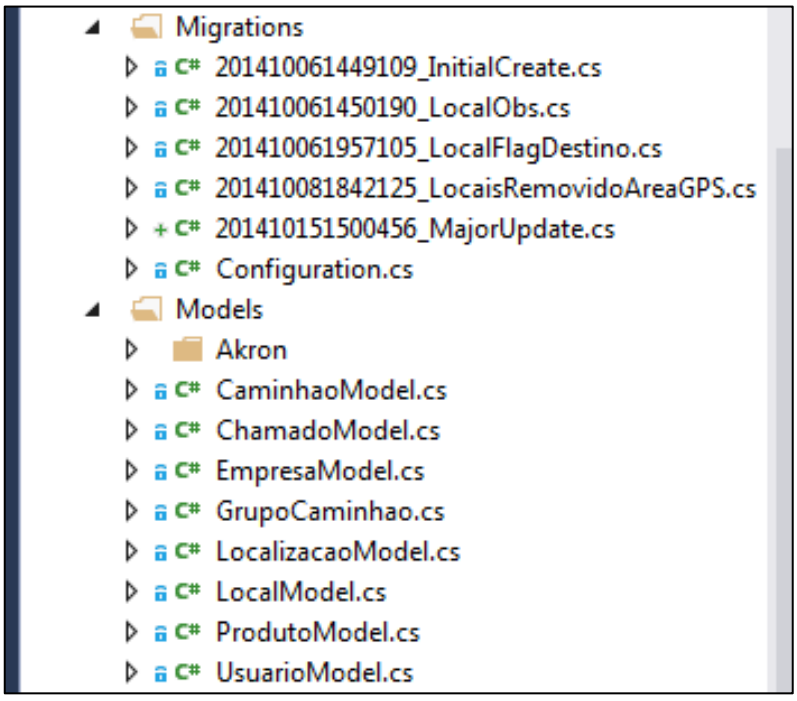
O maior problema provém da necessidade de controlar e executar as atualizações no banco de dados do cliente. Atualizar a aplicação em si é uma tarefa simples, bastando fazer o *deploy* da nova versão no servidor. O banco em produção, no entanto, não pode simplesmente ser substituído pois os dados precisam sem mantidos.

A solução proposta e adotada para solucionar este problema foi utilizar o recurso de *Migrations* disponibilizado pelo Entity Framework, a biblioteca de banco de dados empregada nos projetos .Net. Este permite controlar as modificações na

estrutura/schema do banco de dados que ocorrem durante o desenvolvimento. Qualquer alteração no banco, como adição de tabelas, remoção de campos, permissão de valores NULL, pode automaticamente ser transformada em uma migração que é representada por um trecho de código C#.

Com este controle sabe-se de forma precisa em qual versão se encontra cada banco e, de forma automática, pode-se gerar um *script* SQL que permite migrar a base para o *schema* mais recente do sistema. A Figura 10 é uma imagem dos arquivos do projeto dentro do Visual Studio. A pasta 'Models' contém as classes C# que representam as tabelas do banco de dados, e pasta 'Migrations' contém as migrações geradas automaticamente pelo framework.

Figura 10 – Migrações no C#



Fonte: autor

Executar e gerenciar as migrações envolve alguns comandos simples no *console* do Visual Studio. Maiores detalhes de como executar as migrações estão fora do escopo deste trabalho.

Finalmente, a adoção na empresa deste mecanismo teve êxito. Atualmente as migrações são feitas sempre antes de realizar o *deploy* de um novo *release* do sistema, momento onde é necessário atualizar a estrutura do banco. Isto se tornou imprescindível para a escalabilidade do sistema para vários clientes já que antes

atualizar o banco de dados para a versão mais atual envolvia um processo manual e suscetível a erros.

4.2.4 Resultados

A efetiva utilização do Git demonstrou-se inicialmente como um desafio para ambos desenvolvedores por desconhecerem este tipo de ferramenta, no entanto a adaptação da equipe foi fácil e rápida. Como resultado a empresa optou por adotar de fato o controle de versão. Na prática diária do desenvolvimento do produto, se tornou rotina a sincronização de novos *commits* entre os desenvolver sempre que é necessário compartilhar uma nova funcionalidade.

Um resultado importante também foi a agilidade ganha com o recurso de *merge* para resolução de conflitos. Até então o *merge* de código-fonte era feito de modo manual, com um desenvolvedor copiando os arquivos do outro, efetuando o *merge*, e devolvendo os arquivos para o outro desenvolvedor. Este processo exigia que ambos parassem suas atividades para não ocorrer novas modificações nos arquivos, e também era suscetível a erros pois diversas vezes modificações não eram corretamente copiadas. Com o Visual Studio esta tarefa se tornou mais ágil e segura. Na operação de *push* do Git, o *merge* na maioria das vezes é feito de forma automática, sem necessitar intervenção do programador. Quando ocorrem alterações conflitantes que exigem a decisão do programador de como se dará o *merge*, é fornecido para isso a ferramenta de *merge*, apresentando um comparativo entre as versões conflitantes de arquivos e fornecendo em tempo real o arquivo resultante.

Quanto ao atendimento dos requisitos descritos na Atividade 1, a escolha do Git como sistema de controle de versão permitiu atender os requisitos R1, R2 e R3 pois, como na maioria dos SCV, esses são requisitos essenciais já inclusos. O R4 também foi atendido graças a integração nativa do Visual Studio ao Git.

Para os requisitos R5 e R6, observa-se que o Git, assim como na maioria dos SCV, a princípio não atende este requisito pois é necessário a utilização de um sistema aparte que forneça uma interface Web para visualização do repositório e o sistema integrado de controle de modificações. O atendimento deste requisito será dado através da Atividade 3 do estágio.

4.3 Controle de modificações

Após a adoção do controle de versão, foi possível iniciar a Atividade 3 de escolha e adoção de uma ferramenta para controle de modificações.

4.3.1 Seleção do sistema de controle de modificações

Esta atividade iniciou no dia 01/11/2014 com uma discussão na empresa sobre as opções de ferramentas existentes. A discussão se deteve principalmente entre escolher adotar um serviço na nuvem ou um sistema instalado localmente.

Em serviços on-line de controle de modificações com integração a um SCV, como por exemplo o GitHub e o Bitbucket, obrigatoriamente todo o repositório de código da empresa também deve ser mantido on-line para que haja a integração entre ambos os serviços. Com a discussão, a principal desvantagem que se observou em se usar um serviço on-line é o risco de acesso indevido as informações que são vitais para empresa, como o código fonte dos sistemas. A vantagem, no entanto, é a desnecessidade de precisar instalar e manter um software local para esse propósito.

A decisão que se obteve em consenso foi a de adotar um software instalado localmente. Percebeu-se também esta ser a melhor opção devido ao repositório GIT já estar criado localmente, conforme realizado na Atividade 2, sendo de fácil acesso e maior controle do que um repositório remoto.

Partiu-se então para a pesquisa de opções de ferramentas para o propósito de gerenciamento de modificações integrado ao controle de versão. Percebeu-se que as opções comerciais são extremamente caras, sendo voltadas a suportar projetos grandes, para diversos integrantes e fluxos de trabalhos. Já as opções gratuitas se demonstraram adequadas à necessidade da empresa quanto aos recursos disponibilizados. Identificou-se as seguintes opções de ferramentas, todas elas *open-source*: Bugzilla, Mantis, Redmine e Trac.

O Bugzilla é uma ferramenta para gerenciamento de *bugs*, e não de tarefas, não sendo adequado ao propósito de controlar modificações. O Mantis e o Trac não suportam múltiplos projetos, não atendendo ao requisito R10 definido na Atividade 1. Identificou-se então o Redmine como melhor opção por atender todos os requisitos.

O Redmine (REDMINE, 2014) é uma ferramenta de gerenciamento de projetos, *open source* sob licença GPL, implementada na linguagem Ruby com o framework Rails. Entre as funcionalidades dessa ferramenta, destaca-se:

-Possui tradução para o português;

-Integração com controle de versão GIT e visualização de estatísticas;

-Suporte a múltiplos projetos;

-Suporte a vários bancos de dados, entre eles o SQLServer;

-*Wiki* para colaboração;

-Notificações por e-mail sobre atualizações nas tarefas.

4.3.2 Adoção do Redmine

Primeiramente realizou-se a instalação e configuração da ferramenta. Para a instalação bastou seguir o tutorial disponível no site para um ambiente Windows, sendo instalado na mesma máquina onde estava localizado o repositório GIT central. Para o banco de dados, criou-se uma base do SQLServer para a ferramenta, e também se configurou um sistema de backup periódico da base.

A configuração do Redmine envolveu criar os usuários, criar o projeto do rastreador de frotas e colocar o logo da empresa no cabeçalho do site. Também se personalizou os campos disponíveis para a tarefa:

-Tipo: Funcionalidade ou Defeito

-Título: a atividade desta tarefa

-Descrição: maiores detalhes sobre a tarefa

-Situação: Nova, Rejeitada ou Resolvida

-Prioridade: Baixa, Normal ou Alta

-Atribuída para: responsável por executar ou resolver a tarefa

-Categoria: a qual cliente a tarefa está relacionada

Por fim se importou as tarefas atuais de desenvolvimento mantidas e em uma planilha Excel. O processo foi feito manualmente e contemplou criar 38 tarefas.

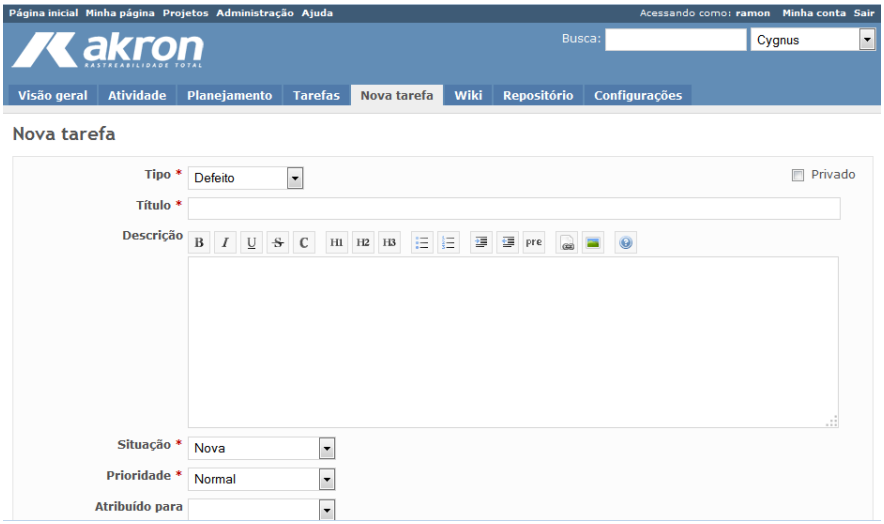
Com a ferramenta instalada e configurada, o próximo passo se deu em liberar a ferramenta para utilização pelos colaboradores da empresa. Para isso, no dia 05/11/2014 se realizou uma exposição com slides e *datashow*, apresentado a ferramenta, seus recursos, um guia de utilização, e o seu propósito de garantir maior controle e transparência sobre as atividades internas da empresa. Ao final se disponibilizou o link de acesso à ferramenta pelo browser e o usuário e senha de cada pessoa.

Obteve-se um *feedback* positivo quanto a utilidade da ferramenta e seu propósito de ser usado para gerenciar as tarefas. No entanto, ressaltaram-se alguns problemas de usabilidade. Cada pessoa relatou possuir uma forma própria de organizar suas tarefas, que incluem métodos mais simples de anotação e, portanto, a nova ferramenta deveria ser simples e de fácil acesso, garantindo sua praticidade de modo a tornar o seu uso habitual.

4.3.3 Fluxo de trabalho no Redmine

Embora o Redmine forneça mecanismos para criar fluxos rígidos para as tarefas, atribuindo papéis e pontos de verificação no andamento de cada uma, conforme necessidade da empresa, um fluxo muito simples foi utilizado. O usuário cria uma nova tarefa a partir da tela mostrada na Figura 11.

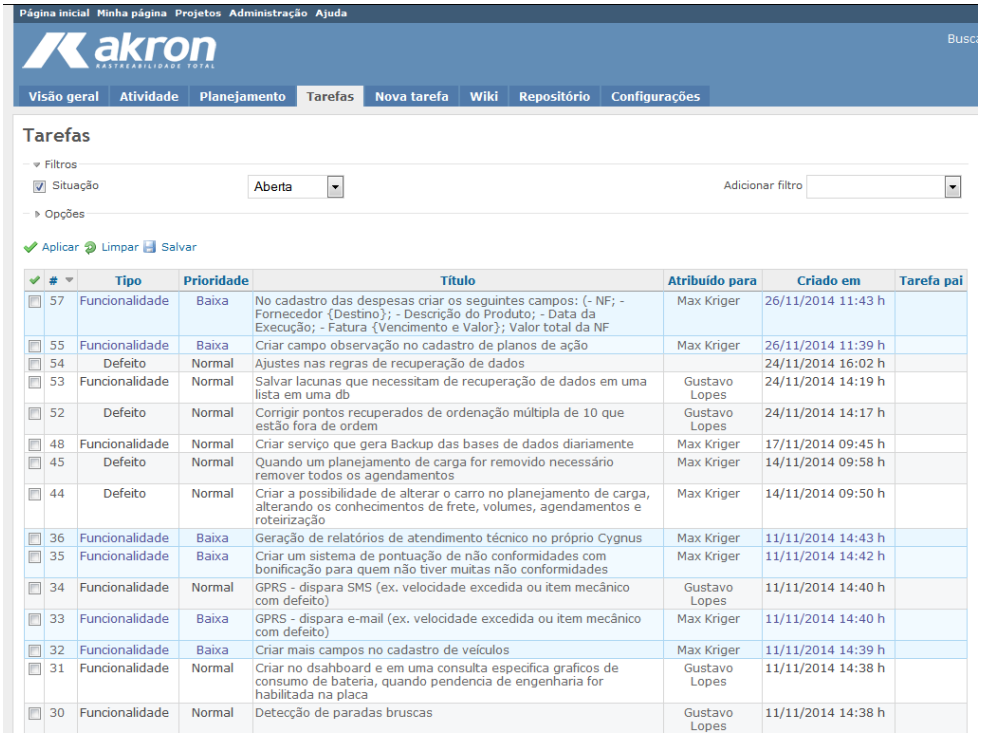
Figura 11 - Criação de tarefa no Redmine



Fonte: autor

Na tela de Tarefas do Redmine (Figura 12) são exibidas todas as tarefas em situação ‘Nova’, ou seja, ainda não concluídas. As tarefas com cor em destaque são as que possuem prioridade mais alta. Pode-se ordená-las por prioridade, por cliente, e também visualizar as tarefas que já foram concluídas.

Figura 12 - Listagem de tarefas do Redmine



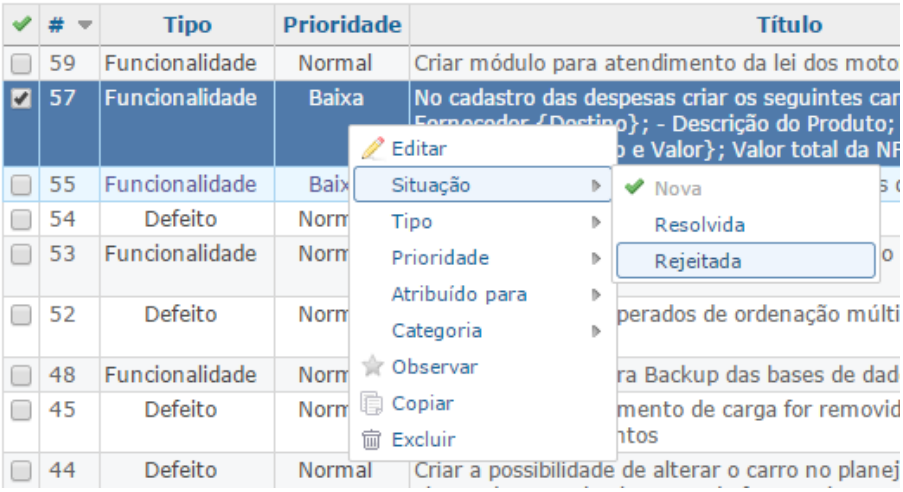
#	Tipo	Prioridade	Título	Atribuído para	Criado em	Tarefa pai
57	Funcionalidade	Baixa	No cadastro das despesas criar os seguintes campos: (- NF; - Fornecedor {Destino}; - Descrição do Produto; - Data da Execução; - Fatura {Vencimento e Valor}; Valor total da NF	Max Kriger	26/11/2014 11:43 h	
55	Funcionalidade	Baixa	Criar campo observação no cadastro de planos de ação	Max Kriger	26/11/2014 11:39 h	
54	Defeito	Normal	Ajustes nas regras de recuperação de dados		24/11/2014 16:02 h	
53	Funcionalidade	Normal	Salvar lacunas que necessitam de recuperação de dados em uma lista em uma db	Gustavo Lopes	24/11/2014 14:19 h	
52	Defeito	Normal	Corrigir pontos recuperados de ordenação múltipla de 10 que estão fora de ordem	Gustavo Lopes	24/11/2014 14:17 h	
48	Funcionalidade	Normal	Criar serviço que gera Backup das bases de dados diariamente	Max Kriger	17/11/2014 09:45 h	
45	Defeito	Normal	Quando um planejamento de carga for removido necessário remover todos os agendamentos	Max Kriger	14/11/2014 09:58 h	
44	Defeito	Normal	Criar a possibilidade de alterar o carro no planejamento de carga, alterando os conhecimentos de frete, volumes, agendamentos e roteirização	Max Kriger	14/11/2014 09:50 h	
36	Funcionalidade	Baixa	Geração de relatórios de atendimento técnico no próprio Cygnus	Max Kriger	11/11/2014 14:43 h	
35	Funcionalidade	Baixa	Criar um sistema de pontuação de não conformidades com bonificação para quem não tiver muitas não conformidades	Max Kriger	11/11/2014 14:42 h	
34	Funcionalidade	Normal	GPRS - dispara SMS (ex. velocidade excedida ou item mecânico com defeito)	Gustavo Lopes	11/11/2014 14:40 h	
33	Funcionalidade	Baixa	GPRS - dispara e-mail (ex. velocidade excedida ou item mecânico com defeito)	Max Kriger	11/11/2014 14:40 h	
32	Funcionalidade	Baixa	Criar mais campos no cadastro de veículos	Max Kriger	11/11/2014 14:39 h	
31	Funcionalidade	Normal	Criar no dsahboard e em uma consulta especifica graficos de consumo de bateria, quando pendencia de engenharia for habilitada na placa	Gustavo Lopes	11/11/2014 14:38 h	
30	Funcionalidade	Normal	Detecção de paradas bruscas	Gustavo Lopes	11/11/2014 14:38 h	

Fonte: autor

As tarefas podem ser constantemente revisadas e suas informações alteradas. O campo Notas da tarefa permite a todos realizar uma discussão sobre o seu andamento.

Por fim, o usuário deve concluir ou rejeitar a tarefa, clicando com o botão direito na linha da tarefa e escolhendo a nova situação (Figura 13).

Figura 13 - Alterando situação da tarefa no Redmine



Fonte: autor

4.3.4 Rastreabilidade das tarefas com *smart commits*

Como definido nos requisito R10, o sistema de controle de modificações deve estar integrado ao SCV, o que é essencial para possibilitar rastrear as alterações nos itens de configuração (código-fonte) responsáveis no atendimento ou criação de uma pendência de modificação.

O Redmine, com sua integração ao GIT, permite relacionar um *commit* de código fonte no repositório a uma tarefa. Esse recurso de *smart commit* permite atualizar as informações da tarefa como percentual de execução, tempo gasto e estado da tarefa, fornecendo um mecanismo de rastreabilidade entre a tarefa e a alterações no código fonte.

Para isso, na mensagem descritiva do *commit* no GIT, o programador deve utilizar a seguinte sintaxe vinculando o *commit* as tarefas:

(ref | close) (#<task_id>)+ [<porcent_of_conclusion>%] [@<spent_time>]

O *ref* e *close* são parâmetros definidos a partir de configurações do Redmine. O *ref* é usado para referenciar uma tarefa a um *commit*, e o *close* é utilizado para automaticamente concluir a tarefa. O símbolo '*#*' seguido do *task_id* (identificador tarefa) é utilizado para relacionar o *commit* à tarefa. Com isso é possível rastrear quando, quem e como a pendência foi resolvida.

4.3.5 Resultados

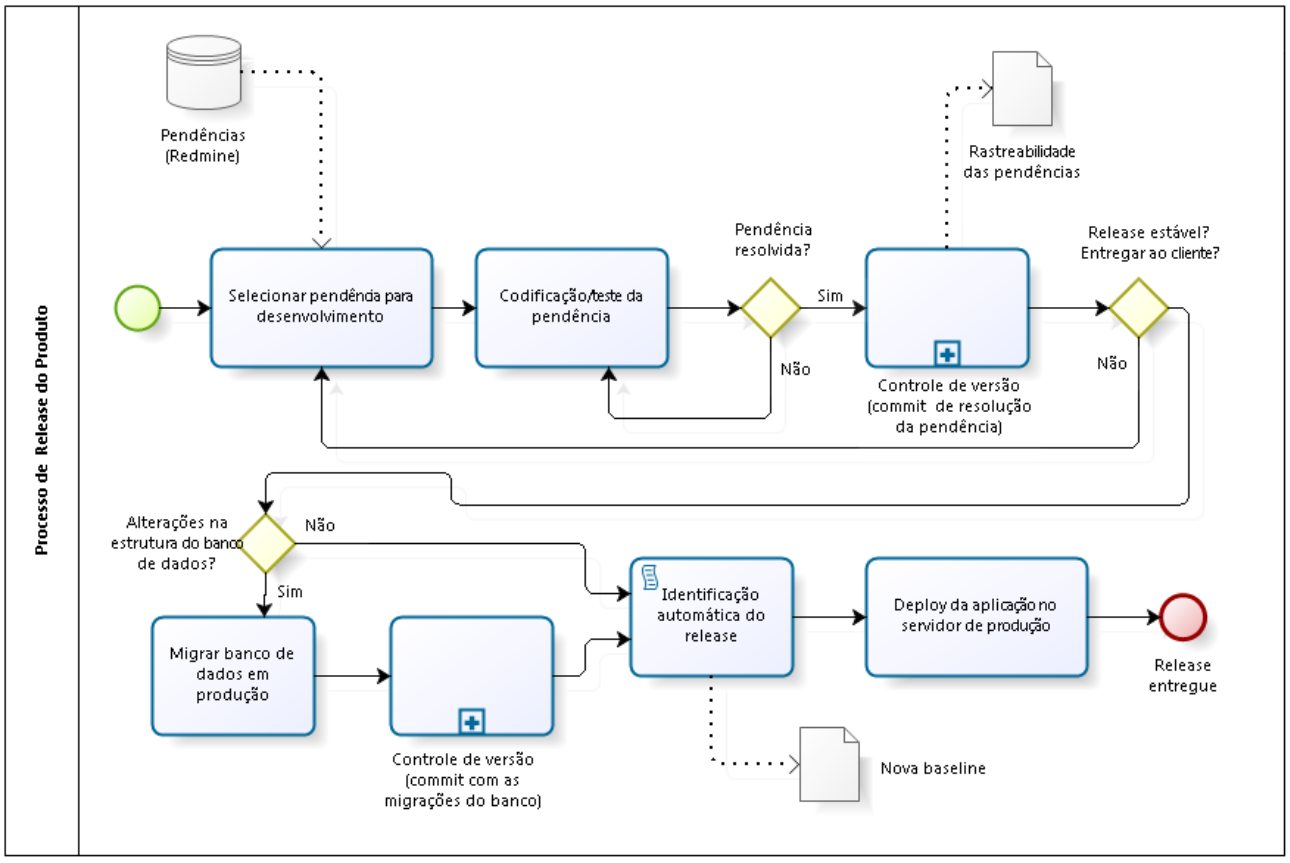
Até o final do estágio, não houve a efetiva utilização da ferramenta pela área de engenharia da empresa. Na área de desenvolvimento, o ambiente do Redmine se tornou de fato o mecanismo para o controle das tarefas de desenvolvimento do produto de gerenciamento de frotas.

Quanto ao atendimento dos requisitos descritos na Atividade 1, o Redmine atendeu os requisitos R8, R9 e R10 satisfatoriamente. Com a funcionalidade de *smart commits* foi possível vincular os *commits* de código efetivamente gerados com as respectivas tarefas no sistema, atendendo assim o requisito R11.

4.4 Descrição do novo processo de desenvolvimento

A Atividade 4 envolveu o desenho e descrição do novo processo utilizando BPMN (*Business Process Modeling Notation*), que é um padrão para a modelagem de processos de negócios. O objetivo do BPMN é prover uma notação rapidamente compreensível por todos os participantes dos processos do negócio (BPMI, 2011).

Figura 14 – Desenho do novo processo



Fonte: autor

O processo desenhado (Figura 14) descreve a geração de um novo *release* do sistema, tal como se estabeleceu através deste trabalho, envolvendo as atividades diárias de desenvolvimento baseadas na resolução das pendências, o controle de versão, culminando com a criação e *deploy* de um novo *release*.

O processo inicia com o programador selecionando uma pendência no Redmine e realizando sua implementação. Quando a pendência é resolvida, o programador deve realizar um *commit* com uma mensagem adequada que conclui a pendência no Redmine, conforme descrito anteriormente. Com isso é possível rastrear quando, quem e como a pendência foi resolvida.

O sub-processo 'Controle de versão' é o processo GIT já apresentado na Figura 5.

Quando há necessidade de entregar ao cliente a resolução de determinadas pendências ou quando se identificou que funcionalidades suficientes foram agregadas ao sistema, se inicia então a atividade de criação de um novo release.

Antes do *deploy* da aplicação, caso a estrutura do banco tenha sido alterada, é necessário migrar o banco de dados em produção, atualizando-o para a nova estrutura. Essa etapa envolve uma série de comandos, conforme descrito anteriormente. A migração resulta em um arquivo C# criado automaticamente que necessita ser versionado através de um novo *commit*.

Ao realizar o *deploy* da aplicação, existe um *script* que automaticamente identifica a aplicação com a data em que foi gerada, o número (*hash*) e mensagem do último *commit* do GIT, e a migração do banco sendo usada. Essas informações são exibidas no rodapé da aplicação Web, como mostrado na Figura 15, e permitem facilmente saber qual revisão do sistema está rodando no cliente.

Figura 15 - Identificação do *release*

Fonte: autor

Portanto, neste processo, uma nova *baseline* é criada sempre que é realizado o *deploy*. A partir da *hash* GIT ou data da aplicação, pode-se acessar o Redmine e verificar quais pendências estão contempladas na versão do sistema. Não se verificou possível na empresa criar um ambiente de homologação onde uma nova *baseline* é gradualmente construída e testada pois a empresa precisa de extrema agilidade na entrega das atualizações. Ao final do processo o novo *release* está rodando no ambiente de produção e entregue ao cliente.

5. Conclusão

A proposta de implantar ferramentas de apoio, visando iniciar a Gestão de Configuração na empresa, foi concluída com êxito. Houve a efetiva utilização das ferramentas e o processo sugerido está hoje plenamente adotado.

Como contribuição, o trabalho trouxe à empresa enormes benefícios. Na parte técnica, o controle de versão e as migrações são agora vitais para administrar as inúmeras versões do sistema em produção. Estes também são vitais para garantir a escalabilidade do sistema de modo a comportar o crescente número de clientes que a empresa vem ganhando. Na parte de organização, o controle de modificações com a ferramenta Redmine demonstrou-se prático, trazendo transparências as atividades e um melhor acompanhamento da evolução do desenvolvimento do software e a rastreabilidade do que foi feito.

Ambas as ferramentas adotadas tiveram custo zero para a empresa, e com isso conclui-se que é possível implementar um processo de GCO através de ferramentas livres.

Como resultados qualitativos destaca-se a agilidade obtida com o GIT para controle de conflitos e merge de código fonte, o que é essencial para suportar o desenvolvimento por mais de uma pessoa. Também, com a utilização do GIT, automaticamente obteve-se uma forma de backup do código-fonte do sistema no repositório central.

Como resultado quantitativo, alguns números importantes foram obtidos ao final do estágio, evidenciando a efetiva utilização das ferramentas propostas. Essas estatísticas abrangem o período de 01/10/2014 até 02/12/2014.

Através do Redmine verificou-se que:

-Defeito: 6 tarefas abertas / 10 total

-Funcionalidade: 29 tarefas abertas / 38 total

Referente ao controle de versão, usando a ferramenta *gitstats* foi possível obter estatísticas sobre o uso do repositório GIT:

-63 *commits*/revisões criadas no repositório GIT

-6501 linhas de código adicionadas;

-3029 linhas de código removidas;

-3472 novas linhas de código.

Verificando o banco de dados e arquivos do projeto, se obteve os seguintes resultados:

-7 clientes e banco de dados em produção, todos controlados por migrações;

-13 migrações/versões do banco de dados foram criadas e auxiliaram o *deploy* das aplicações em produção.

Referências

DORSEY, Terrence. **Source Code Control with Git and Mercurial**. Disponível em: <http://visualstudiomagazine.com/Articles/2014/04/01/Source-Code-Control-with-Git-and-Mercurial.aspx?Page=2>. Acesso em 2/10/2014.

FREITAS, Daniel Tannure Menandro de. **Análise Comparativa entre Sistemas de Controle de Versões**. 2010. 56 f. TCC (Graduação) - Curso de Bacharelado em Ciência da Computação, Universidade Federal de Juiz de Fora, Juiz de Fora, 2010.

OLAUSSON, Mathias et al. **Pro Team Foundation Service**. Springer Link, 2013. Disponível em: <http://link.springer.com/book/10.1007/978-1-4302-5996-1>. Acesso em 15 nov. 2014.

Redmine. Disponível em: <http://www.redmine.org/>. Acesso em: 15 nov. 2014.

SEI. **Capability Maturity Model Integration, v1.2**. 2001. Disponível em <http://www.sei.cmu.edu/library/assets/whitepapers/cmmi-dev_12_portuguese.pdf>. Acesso em: 10 set. 2014.

SOFTEX. **MPS.BR – Melhoria do Processo de Software Brasileiro, Guia de Implementação – Parte 2: Fundamentação para Implementação do Nível F do MR-MPS-SW:2012**. Disponível em: <http://www.softex.br/mps>. Acesso em: 10 set. 2014.

YOSHIDOME, Ewelton Yoshio Chiba. **UMA PROPOSTA DE APOIO SISTEMATIZADO À GERÊNCIA DE CONFIGURAÇÃO DO MPS.BR UTILIZANDO FERRAMENTAS DE SOFTWARE LIVRE.** 2009. 84 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal do Paraná, Belém, 2009.