UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PW-PLANTE: UMA ESTRATÉGIA PARA MELHORIA DE PROCESSO BASEADA EM ATIVIDADES DE PLANEJAMENTO E TESTE

DEYSIANE MATOS SANDE

ORIENTADORA: PROFª. DRª. SANDRA CAMARGO P. F. FABBRI

São Carlos - SP
Julho/2010

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PW-PLANTE: UMA ESTRATÉGIA PARA MELHORIA DE PROCESSO BASEADA EM ATIVIDADES DE PLANEJAMENTO E TESTE

DEYSIANE MATOS SANDE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientadora: Dra. Sandra Camargo Pinto Ferraz Fabbri.

São Carlos - SP
Julho/2010

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S214pe

Sande, Deysiane Matos.
PW-PlanTe : uma estratégia para melhoria de processo baseada em atividades de planejamento e teste / Deysiane Matos Sande. -- São Carlos : UFSCar, 2012.
163 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2010.

1. Software - testes. 2. Planejamento. 3. Melhoria de processo. 4. Métodos ágeis. I. Título.

CDD: 005.14 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"PW-PlanTe: uma estratégia para melhoria de processo baseada em atividades de planejamento e teste"

DEYSIANE MATOS SANDE

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Sandra C. Pinto Ferraz Fabbri
(Orientadora - DC/UFSCar)

Prof. Dr. Valter Vieira de Camargo
(DC/UFSCar)

Profa. Dra. Selma Shn Shimizu Melnikoff
(POLI/USP)

São Carlos
Agosto/2010

Dedico este trabalho aos meus pais que me apoiaram durante todo esse tempo, e mesmo de longe se fizeram presentes me auxiliando em tudo que foi possível.

A eles, meu eterno agradecimento.

AGRADECIMENTO

Inicialmente agradeço a Deus por ter me dado a oportunidade de alcançar tudo que já me foi possível alcançar até o momento. Agradeço por toda orientação e força. Sem Ele e seus ensinamentos esse trabalho não seria possível.

À minha orientadora, Sandra Fabbri, pela excelente orientação e por todos os momentos compartilhados nessa caminhada. Sem Ela, esse trabalho também não seria possível.

Aos meus pais pelo amor incondicional e por todo auxilio que me prestam constantemente.

Aos meus irmãos e toda a minha família por acreditarem em mim e me passarem vibrações positivas para que tudo desse certo.

A todos os companheiros de mestrado que pude compartilhar do convívio durante esse tempo.

Aos amigos, que de longe ou de perto, virtualmente ou fisicamente se fizeram presentes me apoiando sempre.

À amiga Mariana pelo suporte psicológico, companheirismo e socorros nos diversos momentos dessa caminhada.

Aos companheiros Lapesianos Arnaldo, Renan, Dênis, Luiz, Elis, Daniel, Augusto, Vinicius, Fábio, Enrique, André, Anderson, Erick, Guilherme, Fernanda, Juciara e Arlindo por todos os momentos compartilhados e auxílio mútuo.

Às empresas Linkway e NBS que permitiram que pudessem ser realizados os estudos de caso, bem como de todos os seus colaboradores, por todos os momentos de trabalho e também descontração.

Ao CNPQ pelo apoio financeiro.

"Ama sempre, fazendo pelos outros o melhor que possas realizar. Serve sem apego e assim vencerás"

Chico Xavier

RESUMO

Contexto: A implantação de modelos de qualidade de processo em empresas de pequeno porte requer uma adequação para esse tipo de empresa e enfrenta uma maior dificuldade devido às restrições dos diversos tipos de recursos. Para dar suporte à melhoria de processo nesse contexto, no que diz respeito à atividade de planejamento de software, foi definida a estratégia $PCU|_{PSP}$, que está fundamentada na técnica Pontos de Casos de Uso e no *Personal Software Process*. ***Objetivo***: O Objetivo deste trabalho é apresentar a estratégia PW-PlanTe, que é uma evolução da estratégia $PCU|_{PSP}$. A PW-PlanTe generaliza as técnicas a serem adotadas no planejamento, utiliza iterações bem definidas de modo a atender o paradigma de desenvolvimento ágil, e define um processo de teste. Além disso, a PW-PlanTe é apoiada por um conjunto de ferramentas livres que facilitam sua implantação. ***Metodologia***: Dada a aderência da $PCU|_{PSP}$ aos métodos ágeis, essa estratégia foi estudada e generalizada no que diz respeito à parte de planejamento, para se aproximar ainda mais do framework Scrum. Após isso, foi definido o processo de teste para o contexto de uma empresa de pequeno porte e foram identificadas ferramentas livres para dar suporte a todos os passos da estratégia. ***Resultados***: A PW-PlanTe foi aplicada em três sistemas que estavam em desenvolvimento nas empresas de pequeno porte participantes do trabalho. Em dois sistemas, nos quais foi explorado o suporte à atividade de planejamento, os resultados mostraram que a estratégia facilita a determinação de trabalho a ser realizado em cada iteração. Em dois sistemas, nos quais foi explorado o suporte à atividade de teste, vários defeitos foram encontrados com o uso da estratégia e não foram revelados sem a sua aplicação. ***Conclusões***: Com base nos resultados, existem evidências da contribuição da estratégia para a implantação da melhoria de processo de desenvolvimento de software em empresas de pequeno porte, tanto no que diz respeito à atividade de planejamento, como de teste.

Palavras-chave: teste de software, planejamento, controle do planejamento, melhoria de processo, métodos ágeis, empresas de pequeno porte

ABSTRACT

Background: The process quality model implantation on small enterprises requires an adjustment for this type of enterprise and the difficulty to reach this objective is harder due to the restrict amount of different type of resources. To support the process improvement in this context, concerning the software planning activity, the strategy $PCU|_{PSP}$ was defined. It is based on Use Case Points and Personal Software Process. ***Aim***: The objective of this monograph is to present the PW-PlanTe strategy, which is an evolution of the $PCU|_{PSP}$ strategy. PW-PlanTe generalizes the techniques used in the planning, applies well defined iterations aiming to be in conformance with the agile paradigm development, and defines a testing process. In addition, PW-PlanTe is supported by a set of free tools that facilitate its implantation. ***Methodology***: Considering that $PCU|_{PSP}$ is naturally adherent to the agile methods, it was studied and generalized – in relation to the planning issues – aiming to make it even closer to the Scrum framework. After that, the testing process for small enterprises was defined and the free tools were identified aiming to establish a support for the strategy steps application. ***Results***: PW-PlanTe was applied in three systems that were under development by the small enterprises which participated of this research. In two of the four systems where the planning activity support was explored, the results showed that the strategy facilitates the determination of the work to be done in each iteration. In three of the four systems where the testing activity support was explored, various defects were found by the application of the strategy and were not found without its application. ***Conclusion***: Based on these results, evidences can be identified on the contribution of the strategy for the implantation of the software development process improvement in small enterprises, both for the planning activity as well as for the testing activity. Additional studies should be done aiming to make these results more evident.

Keywords: software testing, planning, planning tracking, process improvement, agile method, small companies

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIE – *Arquivo de Interface Externa*

ALI – *Arquivo Lógico Interno*

ALM - *Application Life Cycle Management*

API – *Application Programming Interface*

APF – *Análise por Pontos de Função*

API – *Application Programming Interface*

ASD – *Adaptive Software Development*

CE – *Consultas Externas*

CMM – *Capability Maturity Model*

CMMI – *Capability Maturity Model Intregration*

DSDM – *Dynamic System Development Model*

EE- *Entrada Externa*

FA – *Fatores Ambientais*

FCT – *Fatores de Complexidade Técnica*

FDD – *Feature Driven Development*

HTML – *HyperText Markup Language*

ISO – *International Organization for Standardization*

IFPUG – *International Function Point Users Group*

IW – *Item of Work*

LOC – *Lines of Code*

MPSBR - *Melhoria de Processo do Software Brasileiro*

NDE – *Nível de Esforço*

PCU – *Pontos por Caso de Uso*

PF – *Pontos por Função*

PFN – *Pontos por Função Não Ajustados*

PO – *Product Owner*

PSP – *Personal Software Process*

PW – *Piece of Work*

SDLC – *System Development Life Cycle*

SE – *Saída Externa*

SMTP – *Simple Mail Transfer Protocol*

TDD – *Test Driven Design ou Test Driven Development*

TFD – *Test First Development*

UT – *Unidade de Tempo*

XML – *eXtensible Markup Language*

XP – *eXtreme Programming*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo será apresentado o contexto deste trabalho, as questões que motivaram a sua realização, bem como a organização da dissertação.

1.1 Contexto e Motivação

Desde a década de 90, relatórios como o "*The Chaos Report*" (STANDISH, 1995), já apontavam que, somente uma pequena parcela dos projetos de software existentes podia ser considerada como bem sucedida. Ou seja, tiveram a sua conclusão no prazo determinado, dentro do orçamento previsto e de acordo com os requisitos estipulados pelos usuários.

Sabe-se também, que o crescimento dos níveis de concorrência entre as empresas de desenvolvimento de software fez com que houvesse uma maior preocupação com o sucesso dos projetos realizados e também com a qualidade do software desenvolvido.

Para se obter essa qualidade, é necessário planejar o trabalho a ser feito e gerenciar a produtividade, o esforço, o custo e o tempo de desenvolvimento. Além disso, devem também ser realizadas atividades que possibilitem verificar a qualidade do software desenvolvido, tais como as atividades de garantia da qualidade do software. Enfatizar tais atividades no processo de desenvolvimento do software proporciona uma redução na quantidade de trabalho que necessitará ser refeita (PRESSMAN, 2006).

Dentre as atividades de garantia da qualidade, as atividades de Verificação, Validação e Teste (VV&T) devem ser adotadas durante todo o processo de

desenvolvimento para garantir maior qualidade ao produto de software desenvolvido. Tais atividades, assim como as de planejamento e gerenciamento de software, são explicitamente tratadas nos modelos e normas de qualidade de processo mais conhecidos como CMMI (CMMI, 2006), ISO 12.207 (ISO, 1995), ISO 15504 (ISO, 1998) e MPSBR (MPSBR, 2007) e são essenciais para a implantação desses modelos.

No entanto, tais modelos e normas de qualidade, especificam “o que” deve ser feito para se realizar o planejamento de software e alcançar a qualidade, mas não “como” deve ser feito. Para dar suporte à melhoria de processo nesse contexto, foi definida a estratégia $PCU|_{PSP}$ (SANCHEZ, 2008) que define atividades de planejamento e controle do software desenvolvido por meio da aplicação da técnica Pontos por Caso de Uso (PCU) (KARNER, 1993) e do processo PSP (HUMPHREY, 1995).

Contudo, desde a década de 90 até os dias atuais, o processo de desenvolvimento mudou muito, buscando adaptar-se às novas necessidades do mercado como prazos menores e aumento no tamanho dos sistemas. A qualidade se tornou algo não somente almejado como também essencialmente necessário para as empresas conseguirem se manter no mercado. Assim, visando à melhoria na qualidade de seus produtos e serviços, as organizações buscam cada vez mais alternativas que auxiliem a gerenciar seus projetos e garantir a qualidade do produto desenvolvido.

Nesse novo contexto, o surgimento de novos métodos de desenvolvimento, como os métodos ágeis, introduziu novos conceitos como a entrega rápida de software de qualidade e a flexibilidade no processo de desenvolvimento (BECK, 2001). Essas contribuições foram de extrema importância, principalmente no contexto de pequenas e médias empresas de software, que são compostas por equipes menores, possuem orçamentos reduzidos e, além disso, dependem da entrega dentro dos prazos e da qualidade do produto desenvolvido para se manter no mercado. Entretanto, a falta de definições práticas e de avaliações do uso desses métodos ainda prejudica a sua utilização. Mesmo possuindo fases e recomendações bem definidas, ainda faltam diretrizes práticas e o conhecimento de ferramentas que possam auxiliar na aplicação dos métodos ágeis, tanto no âmbito da gerência e planejamento de projetos, como na realização de atividades de garantia da qualidade do software.

Da mesma forma, no que diz respeito à implantação de modelos de qualidade de processo, principalmente no contexto de empresas de pequeno porte, esse mesmo problema é observado. Além disso, no caso desses modelos, ainda há o agravante da resistência à mudança e da absorção e adaptação de seus requisitos para que seja possível utilizá-los no contexto de tais empresas.

1.2 Objetivo

Com base no contexto e motivação descritos anteriormente, o objetivo deste trabalho é apresentar a Estratégia PW-PlanTe, uma estratégia para melhoria de processo que combina atividades de planejamento e teste de software por meio de passos bem definidos que podem guiar as empresas na execução dessas atividades. Além disso, os passos da estratégia são aderentes ao framework ágil Scrum, o que vem também contemplar a empresa de pequeno porte que hoje procura por melhoria da qualidade de processo e produto com base em um processo de desenvolvimento simplificado e voltado às suas características. A sistemática de funcionamento da PW-PlanTe é apoiada por um conjunto de ferramentas livres, que auxiliam na realização e controle do planejamento, bem como na realização das atividades de testes. Dadas as restrições de recursos de uma empresa de pequeno porte, o objetivo deste trabalho foi também prover uma solução que tivesse um baixo custo, de forma a causar o menor impacto possível, devido às restrições de recursos desse tipo de empresa.

1.3 Metodologia de Execução

O trabalho foi desenvolvido por meio do processo constante de observação, aplicação e levantamento de soluções no contexto de duas empresas de pequeno porte, a Linkway e a NBS. A Linkway vinha adotando a aplicação da estratégia $PCU|_{PSP}$, que já tinha sido desenvolvida no contexto dessa mesma empresa, por um outro trabalho de mestrado do grupo de pesquisa. No entanto, embora essa

estratégia estivesse funcionando, a empresa passou a adotar o framework Scrum, para o seu desenvolvimento e, além disso, queria definir um processo de teste, com apoio de ferramentas, para que essa atividade fosse realizada com maior rigor, para melhorar a qualidade de seus produtos.

O trabalho foi iniciado com o estudo de diferentes técnicas de estimativa de software, que pudessem ser utilizadas para auxiliar no planejamento de software, em especial considerando o paradigma ágil de desenvolvimento. Foram também estudadas alternativas de teste que pudessem ser realizadas no contexto de empresas de pequeno porte com o emprego de poucos recursos. Paralelamente a esse estudo, foi feita uma seleção de ferramentas livres que apóiam o emprego de atividades de planejamento de projeto e garantia da qualidade de software, em especial a atividade de teste.

Dessa forma, foram realizadas melhorias contínuas na Estratégia $PCU|_{PSP}$, modificando primeiramente a aplicação das atividades de planejamento, que anteriormente permitiam somente a aplicação dos Pontos por Casos de Uso (PCU) e do *Personal Software Processs* (PSP), passando a nova versão da estratégia a apoiar também outras formas de estimativa. Com a generalização da estratégia, o esforço foi direcionado à seleção de atividades de teste que pudessem ser empregadas de forma a tornar o processo de garantia da qualidade mais formalizado e sem acréscimo de grande esforço na sua realização.

Paralelamente à generalização do planejamento e à seleção de atividades de teste, foram selecionadas e aplicadas ferramentas que possibilitassem o emprego das atividades de planejamento e teste propostas na nova estratégia.

Ao atingir uma versão muito próxima da versão atual da estratégia aqui proposta, foram realizados quatro estudos de caso que possibilitaram fazer uma primeira avaliação da estratégia. Os estudos foram realizados de forma a avaliar as atividades de planejamento, permitindo identificar contribuições com sua aplicação e também de forma a exemplificar e avaliar a aplicação das atividades de teste.

Os resultados obtidos com os estudos realizados durante a definição da estratégia, relacionados ao contexto do planejamento de software, foram publicados no 12th *International Conference on Enterprise Information Systems* (ICEIS 2010), realizado em Portugal, em junho de 2010.

1.4 Organização do Trabalho

Este trabalho é composto por 7 capítulos organizados da seguinte forma: Neste capítulo, Capítulo 1, foi apresentada a introdução caracterizando-se a área de pesquisa na qual o trabalho se encontra inserido, a motivação de seu desenvolvimento e o objetivo do trabalho. No Capítulo 2, uma vez que um dos objetivos do trabalho foi a generalização da estratégia $PCU|_{PSP}$, apresentam-se essa estratégia e os conceitos empregados na sua definição. No Capítulo 3 são apresentados os conceitos relativos à Garantia da Qualidade de Software, mais especificamente à Verificação, Validação e Teste, utilizados para a seleção de atividades de qualidade agregadas à nova estratégia. Nesse capítulo são discutidas ainda ferramentas livres existentes que dão suporte às atividades de teste, e que foram selecionadas para compor a estratégia. No Capítulo 4, são discutidos conceitos sobre os métodos ágeis, os principais métodos existentes, dando-se maior ênfase ao Scrum, as atividades de planejamento e de VV&T no contexto do paradigma ágil e também ferramentas que dão suporte às atividades nesse contexto. No capítulo 5, parte principal deste trabalho, é apresentada a Estratégia PW-PlanTe, discutindo-se o processo de teste e planejamento definidos e o histórico de seu desenvolvimento. No Capítulo 6 são apresentados os resultados obtidos com os estudos de caso realizados com a aplicação da PW-PlanTe no contexto de planejamento e gerenciamento e de teste de software. Por fim, no Capítulo 7, são apresentadas as contribuições e limitações deste trabalho, bem como os possíveis trabalhos que podem ser realizados a partir deste.

Capítulo 2

QUALIDADE DE SOFTWARE: ATIVIDADES DE PLANEJAMENTO E A MELHORIA DO PROCESSO INDIVIDUAL

Neste capítulo são apresentados conceitos sobre a gerência e o planejamento de software e a importância da coleta de métricas e da derivação de estimativas como forma de possibilitar um planejamento mais preciso. Alguns desses conceitos como o método PCU – Pontos por Caso de Uso – e o PSP – Personal Software Process, são utilizados como base para desenvolvimento da estratégia de planejamento denominada $PCU|_{PSP}$., um dos objetos de estudo deste trabalho.

2.1 Considerações iniciais

Estudos realizados desde meados dos anos 90 apontam que a inabilidade de gerenciar projetos de software é uma das principais causas de problemas no desenvolvimento e entrega de softwares.

O *Standish Group*, em sua pesquisa publicada em 1995, intitulada *The Chaos Report* (STANDISH, 1995) mostra que 31,1% dos projetos são cancelados antes de serem completados; 16,2% dos projetos são entregues dentro do prazo, dentro do orçamento e de acordo com os requisitos levantados inicialmente; e 52,7% dos projetos são entregues, mas fora do prazo e orçamento estabelecidos e com menos funcionalidades do que as levantadas no inicio do projeto. Já em estudo publicado em 2001, intitulado *Extreme Chaos* (STANDISH, 2001) a taxa de projetos

cancelados diminuiu para 49% enquanto que projetos com sucesso aumentou de 16% para 28% como pode ser visualizado na Figura 2.1

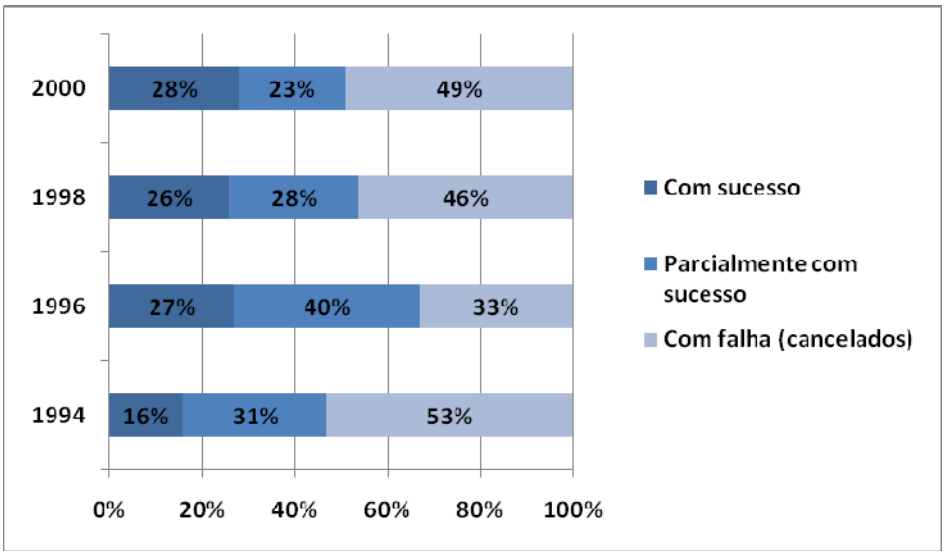


Figura 2.1 - Taxa de sucesso e falhas em projetos de software (adaptado de (STANDISH, 2001))

Essa melhoria se deve a diversos fatores, dentre eles a diminuição dos custos dos projetos, surgimento de melhores ferramentas que auxiliam a monitorar e controlar o progresso e pessoas com melhor preparo em gerenciamento de projetos (STANDISH, 2001).

Em organizações sem processos de software disciplinadamente estabelecidos, o sucesso dos projetos e a qualidade final do produto recaem exclusivamente sobre alguns indivíduos não sendo possível aplicar e manter a qualidade em toda a organização. A melhoria contínua só é conseguida mediante um esforço concentrado de todos os envolvidos; para tanto é imprescindível o desenvolvimento de um processo bem definido com práticas de gerenciamento e planejamento devidamente instituídos.

Diversas normas e modelos de maturidade foram definidos a fim de auxiliar as organizações a alcançar patamares elevados de qualidade. Dentre eles podem-se citar a norma NBR ISO/IEC 12207 (ISO, 1995), a norma ISO/IEC 15504 (ISO, 1998) e os modelos de maturidade como o CMM (PAULK *et al.*, 1993), CMMI (CMMI, 2006) e MPS.BR (MPSBR, 2007), dentre outros.

É possível visualizar nessas normas e modelos, a preocupação com o gerenciamento e planejamento de software sob a forma de orientações de como sair

de um estado caótico para um estado em que os projetos são planejados e minimamente gerenciados.

Construir planos que representem adequadamente o trabalho a ser feito requer a realização de um planejamento elaborado de forma cuidadosa e baseado em informações confiáveis. Dessa forma, é possível estabelecer compromissos que possam ser cumpridos e também o acompanhamento do progresso realizado durante a execução do projeto.

Segundo Humphrey (1995), o planejamento deve ser tratado como uma ferramenta importante que pode ser desenvolvida, praticada e melhorada. Sendo assim, recomenda que a atividade de planejamento de software seja a primeira a ser desenvolvida, englobando, além do próprio planejamento, o estabelecimento de métricas que auxiliem a medir o progresso do projeto, a derivação de estimativas e o controle do planejamento.

Neste capítulo são apresentados os conceitos relacionados à gerência e planejamento de projetos de software. Na Seção 2.2 são discutidos o planejamento de software, bem como as métricas e estimativas existentes que auxiliam na melhoria do planejamento de software. Ainda na Seção 2.2, nas subseções 2.2.2 e 2.2.3 são apresentados a métrica APF (Análise por Pontos de Função) e o método PCU (Pontos por Caso de Uso) respectivamente. Na Seção 2.3 são apresentados os conceitos relativos ao PSP (*Personal Software Process*), que auxilia a melhoria do processo de software por meio da melhoria contínua do individuo, permitindo o controle das estimativas realizadas e o ajuste do planejamento. Na Seção 2.4 é apresentada a estratégia $PCU|_{PSP}$ elaborada com o intuito de auxiliar no planejamento de software no contexto de pequenas empresas. Por fim, na Seção 2.5 são apresentadas as considerações finais relativas ao planejamento de software, à estratégia de planejamento apresentada e aos conceitos discutidos no capítulo.

2.2 Planejamento de Software

Segundo a norma ISO/IEC 15504 (1998), o propósito do gerenciamento de projetos é identificar, estabelecer, coordenar e monitorar as atividades, tarefas e recursos de um projeto. Para tanto, o escopo do projeto deve ser definido, uma

avaliação da viabilidade do projeto deve ser realizada, os recursos e as tarefas disponíveis devem ser dimensionados e os planos para implementação do projeto devem ser elaborados. Além disso, o projeto deve ser continuamente monitorado, e os riscos devem ser continuamente acompanhados e tratados.

Rehesaar e Beames (1998) definem o planejamento de projetos de software como uma atividade que procura tratar de questões como decidir o que fazer, como fazer e quem deverá fazer. Isso ocorre por meio do estabelecimento de objetivos, da estruturação do trabalho em tarefas, do estabelecimento de cronogramas e orçamentos. Além disso, os recursos devem ser alocados de acordo com a distribuição das tarefas, bem como devem ser estabelecidos padrões e selecionadas as ações futuras.

Assim, antes de um projeto ser iniciado, o gerente e a equipe de desenvolvimento de software devem estimar o trabalho a ser feito, os recursos necessários, e o tempo a ser gasto desde o início até o término das ações, controlando os riscos durante todo o projeto.

Entretanto, na prática, as atividades previstas pelo planejamento e gerenciamento de projetos são geralmente negligenciadas. Os planos e cronogramas são realizados de forma puramente intuitiva, os riscos analisados somente quando vêm a acontecer e a organização da equipe e do trabalho é feita de forma imediatista, sem considerar um planejamento em longo prazo.

Obter dados quantitativos que permitam determinar, por exemplo, o tamanho de um projeto de software, pode vir a auxiliar uma organização a sair desse cenário caótico para um patamar em que o mínimo de planejamento possa ser realizado.

O tamanho de um projeto pode estar relacionado à medida da quantidade de linhas de código que ele provavelmente terá ou a um valor mais abstrato que permita visualizar a complexidade de implementação desse software. O grau de complexidade pode ser conseguido por meio da coleta de medidas e da aplicação de métricas que permitam derivar estimativas de tamanho, custo e tempo.

Nas subseções seguintes serão discutidos os conceitos de métricas e estimativas, e serão abordados a métrica APF e o método PCU, que auxiliam no cálculo da complexidade, permitindo caracterizar o tamanho do sistema a ser desenvolvido.

2.2.1 Métricas e Estimativas

As métricas são medidas quantitativas que permitem determinar a eficácia do processo de software e dos projetos realizados (PRESSMAN, 2006). Elas permitem avaliar o estado de um determinado projeto por meio da análise de medidas coletadas durante o seu desenvolvimento, permitindo que os riscos possam ser acompanhados e minimizados. Desse modo, o trabalho realizado pode ser medido e o trabalho a realizar bem como a qualidade do produto podem ser continuamente acompanhados.

As medidas podem ser obtidas de métricas diretas aplicadas durante o processo de desenvolvimento como quantidade de linhas de código (LOC), quantidade de defeitos, custo de desenvolvimento, esforço, etc. Elas também podem ser obtidas de métricas indiretas que avaliam qualidade, confiabilidade, complexidade, manutenibilidade, eficiência, dentre outras. Além disso, as métricas podem ser classificadas em métricas de produtividade, de qualidade, técnicas; podem, ainda, ser orientadas ao tamanho, à função, às pessoas, aos pontos por caso de uso, etc.

A partir das métricas coletadas é possível realizar a derivação de estimativas de custo, esforço e tempo, permitindo a construção de cronogramas mais ajustados. No decorrer do projeto, as estimativas podem ser comparadas às realizadas inicialmente para monitorar e controlar o progresso do projeto, evitando atrasos, minimizando riscos e avaliando a qualidade do produto.

A estimativa de projetos de software permite que se possa determinar quanto será gasto com desenvolvimento, qual o esforço necessário, a quantidade ideal de recursos e quanto tempo irá levar para que o sistema almejado possa ser desenvolvido. Entretanto, não é possível desenvolver uma estimativa significativa sem a delimitação precisa do escopo do produto; portanto, essa é a primeira ação do projeto, após a qual o problema pode ser dividido em uma série de problemas menores e cada um deles pode ser estimado utilizando dados históricos ou mesmo experiência anterior (PRESSMAN, 2006).

Para conseguir estimativas que possuam um maior grau de confiança e graus de riscos aceitáveis, é possível adiar a sua realização até que haja uma maior compreensão sobre o projeto; outra forma consiste em derivar as estimativas, aplicando técnicas de decomposição e/ou modelos empíricos.

As técnicas de decomposição permitem dividir um problema em frações menores para que possam ser melhor estimadas. Já os modelos empíricos usam fórmulas derivadas empiricamente para prever o esforço como uma função de uma métrica, por exemplo, LOC (Linhas de código) ou PF (Pontos por Função).

2.2.2 Pontos por Função – PF

O Pontos por Função (PF) foi criado em 1979 por Allan Albrecht (ALBRECHT,1979), sendo posteriormente refinada pelo *International Function Point Users Group*[1](IFPUG). Essa métrica permite uma contagem indicativa do tamanho do software ainda no início do processo de desenvolvimento, analisando os requisitos do projeto, mas sem necessariamente conhecer em detalhes o modelo de dados. De fato a métrica determina a complexidade de cinco componentes lógicos, a qual, para efeitos práticos, é associada ao tamanho do software.

Usando dados históricos, os valores conseguidos por meio da aplicação da APF podem ser utilizados para estimar o custo ou esforço necessários para o desenvolvimento de um sistema, bem como prever o número de erros, de componentes e de linhas de código a serem implementadas (PRESSMAN, 2006).

Os Pontos por Função são derivados da contagem das funções existentes observadas nos requisitos funcionais do sistema e por meio da avaliação de características não-funcionais como desempenho, usabilidade, reuso, segurança, etc.

A caracterização dos Pontos por Função inicia-se com a contagem de 5 componentes lógicos derivados do domínio da aplicação: Arquivo Lógico Interno (ALI), Arquivo de Interface Externa (AIE), Entradas Externas (EE), Saídas Externas (SE) e Consultas Externas (CE). A partir da contagem desses elementos pode ser obtido o valor do ponto de função não ajustado. Esse valor, após a análise dos requisitos não-funcionais, será ajustado por meio da análise das 14 Características Gerais do Sistema (CGS) que a APF define, fornecendo um valor mais preciso denominado pontos por função ajustados.

A contagem dos componentes lógicos ocorre por meio do levantamento das funções dos tipos dado e transação. As funções tipo dado são representadas pelos

[1] http://www.ifpug.org/

ALIs e AIEs, que representam um grupo de dados logicamente relacionados ou informações de controle. Nos ALIs os dados mantidos estão dentro da fronteira da aplicação, enquanto que nos AIEs os dados pertencem à fronteira de outra aplicação. As funções tipo transação são representadas pelas EEs que permitem a entrada de dados de fora para dentro da fronteira da aplicação; pelas SEs que fornecem informações de dentro para fora da aplicação, por meio da realização de algum cálculo ou computação sobre estas informações; e pelas CEs que fornecem dados ou informações para fora da fronteira da aplicação sem a realização de cálculos sobre estas informações.

Após a coleta desses dados, um valor de complexidade é atribuído a cada uma das funções tipo dado e transação, que serão classificadas em grau Simples, Médio ou Complexo.

Para realizar o cálculo dos pontos por função não ajustado (PFN) é necessário multiplicar cada componente lógico pela complexidade atribuída a ele e somar os valores obtidos nessa multiplicação.

Este valor de pontos por função não ajustado pode ser utilizado para derivar estimativas, entretanto, é recomendo o seu ajuste por meio da análise das Características Gerais do Sistema. A cada uma das características é atribuído um peso que varia de 0 (irrelevante) a 5 (relevante) permitindo adaptar o valor de PF encontrado anteriormente às características relevantes do sistema. Esse ajuste pode ser realizado por meio do uso da fórmula (1), na qual F*i* (i = 1 a 14) representa os valores estabelecidos pela métrica para o ajuste de complexidade:

$$\mathbf{PF = contagem\ total\ de\ PFNs\ x\ [0{,}65 + 0{,}01\ x \sum (F\mathit{i})} \qquad \mathit{(1)}$$

Por possuir uma contagem baseada não em medidas diretas, mas em dados subjetivos, a APF assim como outras métricas subjetivas recebe muitas criticas; entretanto uma de suas vantagens é que ela pode ser aplicada independentemente da linguagem de programação empregada para desenvolvimento.

Além da APF, outros métodos foram propostos na literatura com o objetivo de tornar mais simples a derivação de estimativas que auxiliem no planejamento. Na próxima seção será discutido o método Pontos por Caso de Uso, elaborado por Karner (KARNER, 1993) que, assim como a APF, permite uma contagem

independente de linguagem de programação e que pode ser aplicada também no início do desenvolvimento de um projeto.

2.2.3 Pontos por Caso de Uso - PCU

O método Pontos por Caso de Uso (PCU) é utilizado para facilitar a estimativa de esforço e custo na construção de sistemas de software. O PCU é um método proposto por Gustav Karner (1993), que se baseou na métrica PF (Análise por Pontos de Função) desenvolvida por Allan Albrecht, apresentada na subseção 2.2.2. Calculam-se as estimativas levando-se em consideração a complexidade dos Casos de Uso que compõem o sistema e dos Atores envolvidos; também são levados em consideração os Fatores de Complexidade Técnica (FCT) relacionados aos requisitos não-funcionais e os Fatores Ambientais (FA), relacionados ao nível de competência da equipe responsável pelo sistema (KARNER,1993).

O cálculo do total de Pontos por Caso de Uso é realizado por meio da seguinte fórmula (2):

PCU = Pontos por Caso de Uso não Ajustados * FCT * FA ***(2)***

O valor Pontos por Caso de Uso não Ajustados corresponde à soma das complexidades dos Casos de Uso com as complexidades dos Atores. A complexidade dos Atores e dos Casos de Uso classifica-se em Simples, Média e Complexa, como pode ser visto nas Tabela 2.1 e Tabela 2.2.

Tabela 2.1 – Complexidade dos Atores (adaptado de (KARNER, 1993))

Tipo de Ator	Descrição	Peso
Simples	Outro software acessado por meio de uma API de programação	1
Médio	Outro software interagindo por meio de um protocolo de comunicação como TCP/IP ou FTP	2
Complexo	Um usuário interagindo por meio de uma interface gráfica	3

Tabela 2.2 – Complexidade dos Casos de Uso (adaptado de (KARNER, 1993))

Tipo de Caso de Uso	Nº de Transações	Nº de Classes	Peso
Simples	Até 3	Até 5	5
Médio	4 a 7	6 a 10	10
Complexo	7 ou mais	10 ou mais	15

Os Fatores de Complexidade Técnica e Ambiental são utilizados para ajustar o valor final Pontos por Caso de Uso. Esses fatores possuem pesos pré-determinados que podem ser vistos nas Tabela 2.3 e Tabela 2.4. Para cada um dos fatores deve ser atribuído um nível de influência, que varia de zero (irrelevante) a cinco (essencial). Se o fator não for essencial nem irrelevante e possuir uma influência normal para o sistema, deve-se atribuir a ele o valor três. Caso todos os fatores tenham esse valor intermediário, o valor final será aproximadamente um e não irá influenciar no ajuste dos Pontos por Casos de Uso.

Tabela 2.3 – Fatores de Complexidade Técnica (FCT) (adaptado de (KARNER, 1993))

FCT	Descrição	Peso
F1	Sistema distribuído	2
F2	Tempo de Resposta	1
F3	Eficiência	1
F4	Processamento complexo	1
F5	Código reusável	1
F6	Facilidade de instalação	0,5
F7	Facilidade de uso	0,5
F8	Portabilidade	2
F9	Facilidade de mudança	1
F10	Concorrência	1
F11	Recursos de segurança	1
F12	Acessível por terceiros	1
F13	Requer treinamento especial	1

Tabela 2.4 – Fatores Ambientais (FA) (adaptado de (KARNER, 1993))

FA	Descrição	Peso
E1	Familiaridade com o processo de desenvolvimento.	1,5
E2	Desenvolvedores em meio expediente.	-1
E3	Presença de analistas experientes	0,5
E4	Experiência com a aplicação em desenvolvimento.	0,5
E5	Experiência em Orientação a Objetos.	1
E6	Motivação	1
E7	Dificuldade com a linguagem de programação	-1
E8	Requisitos estáveis	2

Os cálculos dos Fatores de Complexidade Técnica e dos Fatores Ambientais são efetuados por meio das seguintes fórmulas (3) e (4):

$$FCT = 0,6 + 0,01\sum_{i-1}^{13} F_i * Influência_i \quad (3)$$

$$FA = 1,4 - 0,03\sum_{i-1}^{8} E_i * Influência_i \quad (4)$$

nas quais F_i e E_i são, respectivamente, os pesos de cada FCT e FA, e o valor $Influência_i$ é o valor atribuído pelo desenvolvedor e deve estar entre zero e cinco.

Segundo Karner (1993), o nível de esforço para cada Ponto por Caso de Uso é de 20 horas. Dessa forma, se para um determinado sistema obtivermos um valor

total de PCU hipotético de 30 pontos por Caso de Uso, considerando-se o valor de nível de esforço (NDE) igual a 20 proposto por Karner, o tempo total de desenvolvimento do sistema seria obtido multiplicando-se o PCU pelo NDE, constituindo um total de 600 horas.

O PCU proposto por Karner está entre os principais métodos formulados para facilitar a estimativa de esforço na construção de um sistema. Seu cálculo auxilia a prever a complexidade de um sistema de software, caracterizando, indiretamente, o tamanho do software a ser implementado. Entretanto, para se obter resultados mais concretos, o método PCU deve ser adequado à realidade de cada empresa. Essa adequação pode se dar, por exemplo, por meio do ajuste do esforço empregado para o desenvolvimento dos pontos encontrados para cada sistema.

A aplicação de métricas, como a APF, ou métodos, como o PCU, possibilita a determinação de valores que representam o tamanho ou complexidade de um sistema. Entretanto, analisados isoladamente pouco podem dizer sobre o sistema a ser desenvolvido. É necessário derivar estimativas dessas medidas obtidas e adaptar esses valores à realidade de cada empresa.

Na Seção 2.3 será apresentado o PSP, modelo de processo de melhoria pessoal, que, além de auxiliar na melhoria pessoal do indivíduo possibilita o ajuste dos valores encontrados com o APF e o PCU, caso estes conceitos estejam sendo utilizados.

2.3 PSP – Personal Software Process

O *Personal Software Process* (PSP) é um processo de melhoria pessoal utilizado para auxiliar na previsibilidade, na produtividade, bem como na qualidade do desempenho do desenvolvedor e do produto desenvolvido. Seus objetivos principais, portanto, são de prover melhorias na estimativa de projetos e na qualidade do produto, por meio da coleta de tempo, da previsão do tamanho do software a ser desenvolvido e da coleta de defeitos (HUMPHREY, 1995; JOHNSON *et al.*, 2003).

O PSP baseia-se no conceito de Garantia da Qualidade possibilitando uma menor ocorrência de defeitos durante o ciclo de desenvolvimento de um produto.

Para tanto, define cinco atividades que permitem, aos engenheiros de software, de maneira individual, melhorar continuamente seus processos pessoais por meio da aplicação de técnicas de controle de processo. Por meio do controle dessas atividades o desenvolvedor pode determinar quais métodos e técnicas são mais eficazes em seu modo de trabalho.

As cinco atividades de desenvolvimento de software são divididas em Planejamento, Desenvolvimento (Projeto, Codificação e Testes) e Análise Final. Assume-se que antes do Planejamento, o levantamento de requisitos tenha sido efetuado, servindo como entrada à atividade de Planejamento. Durante a atividade de Planejamento, são realizadas as estimativas do trabalho a ser desenvolvido para o qual o desenvolvedor deve estimar valores que se aproximem ao máximo dos futuros dados reais. No Desenvolvimento, o desenvolvedor despende a maior parte do tempo efetivamente construindo o software, projetando, codificando e testando. Por fim, a Análise Final é utilizada para levantar informações a respeito de todo o trabalho realizado nas fases anteriores e estabelecer um comparativo entre o que foi planejado e estimado com o que foi efetivamente executado (HUMPHREY, 1995).

O PSP divide-se em sete níveis, como mostrado na Figura 2.2. Cada nível possui objetivos que devem ser alcançados auxiliando o desenvolvedor a compreender melhor a qualidade do seu processo e do produto construído. Aprendizados como quanto tempo é realmente gasto para a realização de suas atividades, o número e tipo de defeitos encontrados, a padronização e revisão do código, são obtidos em cada um dos níveis.

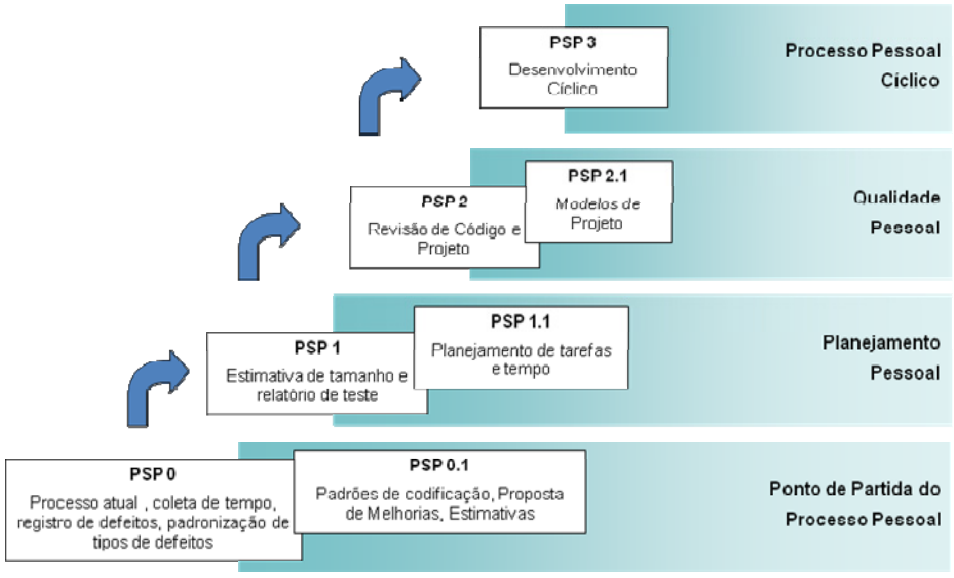


Figura 2.2 - Estrutura do PSP (adaptado de (HUMPHREY, 1995))

O suporte ao PSP se dá por meio da utilização de diversas ferramentas que têm como objetivo tornar a aplicação do modelo de processo mais fácil e prática. Elas são divididas em ferramentas de 1ª, 2ª e 3ª geração.

As ferramentas de 1ª geração foram propostas juntamente com o PSP por Watts Humphrey (1995) e consistem em formulários impressos de preenchimento manual. Entretanto, o tempo gasto pelo desenvolvedor com o preenchimento desses formulários resultava em grande esforço e desmotivava a aplicação do PSP. Além do tempo, o fato dos dados serem coletados manualmente implicava em problemas de qualidade dos dados, afetando consideravelmente as análises elaboradas dentro do contexto do PSP.

A partir dessas constatações, ferramentas computacionais como o *Leap*, *PSP Studio* e *Process Dashboard* (JOHNSON *et al.*, 2003), conhecidas como de 2ª geração, foram desenvolvidas introduzindo facilidades como construção automática de gráficos, base de dados históricos consistentes e de fácil pesquisa, validação dos dados coletados, e inúmeros cálculos realizados automaticamente. Entretanto, a constante troca de contextos, entre o trabalho desenvolvido e o registro das atividades realizadas, fazia com que os desenvolvedores abandonassem o modelo de processo assim que se viam em um ambiente em que o mesmo não era forçosamente aplicado (MONTEBELO, 2008).

De maneira a tornar a coleta de informações automática, sem esforço para o desenvolvedor, no sentido de preencher formulários, ou mesmo apertar botões para iniciar ou encerrar a coleta, surgiram as ferramentas de 3ª geração, caracterizadas por serem agentes de software inteligentes, que podem ser acopladas às ferramentas usadas no desenvolvimento de software. Esses agentes coletam métricas diretamente do ambiente de desenvolvimento, sem a necessidade de intervenção direta do desenvolvedor. Entretanto, nem todas as atividades podem ser completamente automatizadas, pois em alguns casos a intervenção humana para julgar a atividade ainda é necessária. Dessa forma, ferramentas de 3ª geração, como a *Hackystat* (HACKYSTAT, 2010), mudam a natureza de coleta de métricas e, portanto, não conseguem apoiar todas as atividades do PSP. Entretanto, seu uso acompanhado de ferramentas de 2ª geração podem vir a estabelecer o cenário ideal para a aplicação do PSP.

O controle de tempo realizado com o uso do PSP permite que o desenvolvedor adquira um conhecimento mais preciso sobre a sua produtividade, permitindo estimar com mais precisão o trabalho que venha a desenvolver. Dessa forma, é possível obter medidas que permitam ajustar o planejamento realizado em determinado projeto.

2.4 Estratégias de Planejamento

Como dito anteriormente, um dos aspectos fundamentais do planejamento e gerenciamento de projetos é a previsão de quanto tempo o desenvolvimento de um projeto irá durar. Estabelecer estimativas de prazos e custos mais realistas se torna um problema muito mais grave no contexto de empresas de pequeno porte que lidam diariamente com a pressão do mercado para construir sistemas de qualidade com prazos restritos.

No sentido de obter estimativas mais precisas, combinando planejamento e controle, de forma a manter o planejamento constantemente ajustado, Sanchez (2008) propôs a Estratégia de Ajuste por Pontos por Caso de Uso Baseada no PSP ($PCU|_{PSP}$). Essa estratégia foi construída com base no aprendizado obtido na empresa de desenvolvimento de software Linkway, pelo uso constante da técnica Pontos por Caso de Uso – PCU (KARNER, 1993) e da utilização do Personal Software Process - PSP (HUMPHREY, 1995).

A Linkway é uma empresa de pequeno porte localizada na cidade São Carlos que busca constantemente a melhoria do processo de software. Essa preocupação fez com que a empresa adotasse o uso do PSP. Assim, a equipe de desenvolvedores usa as principais práticas do PSP 1.1, relacionadas à estimativa de projeto, juntamente com a ferramenta *Process Dashboard* (DASHBOARD, 2010) que dá suporte ao PSP.

A estratégia $PCU|_{PSP}$ consiste na combinação do PSP com o PCU, sendo que o PCU fornece as estimativas de tamanho (complexidade) e de tempo para a codificação do software, enquanto o PSP determina as bases disciplinadas de um processo pessoal de desenvolvimento que visa à melhoria da qualidade e da produtividade (SANCHEZ, 2008).

A estratégia PCU|$_{PSP}$ apresentada na Figura 2.3 é composta de 9 etapas agrupadas em dois grandes blocos de atividades que são executados constantemente e que se completam, um de planejamento e outro de controle; por meio do feedback das atividades executadas nestes blocos, o planejamento é constantemente ajustado. O bloco de planejamento permite que estimativas adequadas de tamanho, prazo e custos sejam determinadas antes do desenvolvimento do sistema. O bloco de controle verifica se o planejamento estabelecido está sendo obedecido e, caso não esteja, determina os motivos desse fato.

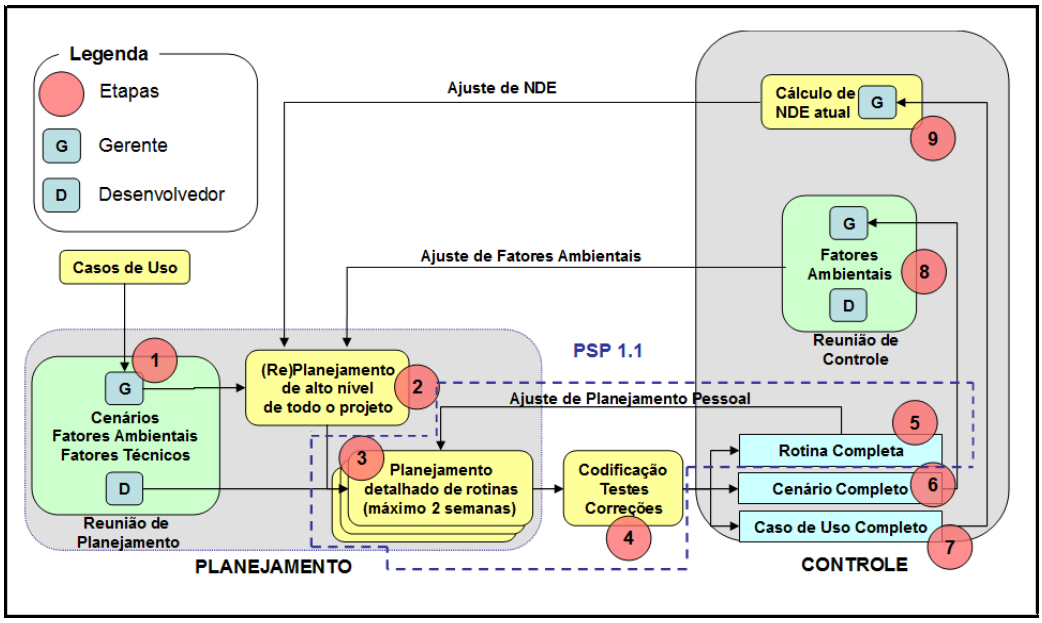


Figura 2.3 - Estratégia PCU|$_{PSP}$ (SANCHEZ, 2008)

Na etapa 1 o Gerente realiza, juntamente com os desenvolvedores, a Reunião de Planejamento em que são discutidos todos os cenários dos Casos de Uso existentes. Nessa reunião é determinada a complexidade dos Casos de Uso, os níveis de influência dos Fatores de Complexidade Técnica e níveis dos Fatores Ambientais.

Na etapa 2 o Gerente realiza o Planejamento de alto nível de todo o projeto baseado no consenso obtido na etapa anterior, sobre os casos de uso. Nessa etapa, através da aplicação do método PCU é possível ter uma estimativa do tamanho do sistema por meio da soma dos Pontos por Caso de Uso. Para determinar o custo e o tempo de desenvolvimento, devem ser multiplicados os PCU atribuídos a cada

desenvolvedor pela variável de Nível de Esforço (NDE) definido individualmente utilizando o PSP.

Na etapa 3 o Planejamento de alto nível da etapa 2 é usado para montar o planejamento detalhado de cada cenário. Os casos de uso são decompostos em cenários sendo que cada cenário pode ser decomposto em rotinas menores. Cada Desenvolvedor deve realizar o planejamento de cada uma das rotinas, de acordo com o PSP Nível 1.1, incluindo um cronograma detalhado de desenvolvimento limitado a duas semanas.

Na etapa 4 o desenvolvedor inicia a atividade de desenvolvimento incluindo codificação, testes e correções. Ao finalizar o ciclo de desenvolvimento de cada rotina inicia-se a etapa 5. Nessa etapa, chamada no PSP de postmortem, o desenvolvedor analisa seu desempenho em comparação ao desempenho estimado durante o planejamento realizado na etapa 3.

Durante a etapa 6 o Desenvolvedor, ao finalizar um conjunto de rotinas que caracteriza um cenário, deve informar esse fato ao Gerente que, por sua vez, deve comparar o desempenho real do desenvolvedor com o desempenho especificado no planejamento de alto nível. Ao concluir a etapa 6, caso haja grande diferença entre o desempenho estimado e realizado, o fluxo de execução muda para a etapa 8, onde deve haver uma reunião de Controle a fim de encontrar possíveis causas e meios de resolver o problema. Os Fatores Ambientais são ajustados e o próximo cenário é selecionado para execução.

Na etapa 7, quando todos os cenários de um Caso de Uso foram desenvolvidos o gerente deve comparar o esforço estimado para o desenvolvimento desse Caso de Uso com o tempo real gasto para tal atividade. Dessa forma, na etapa 9, o NDE de cada desenvolvedor pode ser ajustado logo no primeiro Caso de Uso concluído, para que seja possível refletir sobre a relação entre Pontos por Casos de Uso e o esforço em horas/homens, para cada desenvolvedor individualmente. Para realizar este ajuste é necessário dividir o tempo real gasto, fornecido pelo PSP após a conclusão do Caso de Uso, pela soma dos Pontos dos Casos de Uso já desenvolvidos pelo desenvolvedor. Este mecanismo deve ser repetido sempre que um Caso de Uso seja finalizado, ou seja, acumulam-se o tempo gasto e os Pontos por Caso de Uso já realizados e divide-se o primeiro pelo segundo.

Para se aplicar a estratégia PCU|$_{PSP}$ a uma equipe com vários Desenvolvedores, o gerente deve montar o cronograma geral do projeto aplicando a Estratégia PCU|$_{PSP}$ para cada desenvolvedor.

Com as informações geradas pelo uso do PSP o gerente pode avaliar o desempenho dos desenvolvedores e ajustar o planejamento do software. As informações utilizadas pelo gerente são derivadas do uso do PSP até o nível 1.1.

Segundo Sanchez (2008), a implantação do PSP deve se dar de forma gradual, de acordo com os níveis estabelecidos no processo. Assim, a primeira coisa a ser feita é implantar o Processo de Medição Pessoal, que comporta os níveis 0 e 0.1 do PSP. Com isso, o desenvolvedor se habitua a registrar os tempos e passa a ter um maior conhecimento de seu próprio processo de desenvolvimento.

No nível PSP 0, são executados o Processo Atual (Análise, Projeto, Codificação, Compilação, Testes e Postmortem), o Registro de Tempos, o Registro de Defeitos e o Padrão de Tipos de Defeitos. Após o completo entendimento do PSP 0, o gerente pode mudar para o nível PSP 0.1 em que as atividades padrão de Codificação, Medida de Tamanho e PIP (*Process Improvement Proposals*) são acrescentadas ao cotidiano do desenvolvedor.

Após a incorporação dos níveis PSP 0 e PSP 0.1 Sanchez sugere ainda que o gerente continue a motivar a utilização dos outros níveis do PSP, como o PSP 1.0 e o PSP 1.1. No PSP 1.0 o desenvolvedor realiza as atividades de Estimativa de Tamanho que equivale no PCU|$_{PSP}$ à atividade de planejamento das rotinas dos Casos de Uso, realizada na Etapa 3. É nessa etapa também que os requisitos do PSP 1.1 são atendidos e o desenvolvedor realiza o Planejamento das Tarefas e o Cronograma, de forma que o desenvolvedor faz as estimativas do tempo para o desenvolvimento de cada rotina de um Caso de Uso e planeja as tarefas a executar. Baseando-se no planejamento de cada rotina o cronograma é montado e o gerente pode avaliar se o cronograma e o tempo alocados pelo desenvolvedor estão condizentes com a realidade.

Dessa forma, a estratégia PCU|$_{PSP}$ apresentada por Sanchez, provê auxílio à gerência de projetos, baseando-se no ajuste dos Pontos por Caso de Uso com dados provenientes do uso do PSP. Essa estratégia aborda tanto o planejamento obtido com a aplicação do método PCU, como o tempo gasto com o desenvolvimento e o acompanhamento do progresso do projeto, obtido com o suporte do PSP. Em especial, o PSP supre a deficiência do PCU em estimular a

melhoria pessoal do desenvolvedor para que ele aprenda a estabelecer processos mais eficientes. A estratégia propõe também a coleta de dados estatísticos existentes nas fases de desenvolvimento de software e provê cobertura ao modelo de processo PSP até o nível 1.1. Permanece a lacuna de possibilitar suporte às fases posteriores do PSP e ao ciclo completo de detecção e remoção de defeitos existente no PSP, bem como a coleta de dados relativos a tempo e custo relacionados a esses defeitos.

2.5 Considerações Finais

Neste capítulo foram apresentados alguns conceitos relacionados à gerência e planejamento de projetos de software e a necessidade de realizar planejamentos mais ajustados à realidade de cada empresa. Para conseguir efetivar esses ajustes é necessário o estabelecimento de métricas que permitam a construção de uma base de dados histórica, de forma que possam ser derivadas estimativas de custo, tempo e esforço.

Foram comentadas as métricas clássicas de Pontos por Função e Pontos por Casos de Uso, esta última utilizada em uma estratégia de planejamento desenvolvida por outro trabalho de mestrado do grupo de pesquisa, a qual foi usada como base para o desenvolvimento do trabalho aqui proposto. Essa estratégia, denominada $PCU|_{PSP}$ tem por objetivo apoiar o planejamento do desenvolvimento de software por meio do uso conjunto do método Pontos por Casos de Uso (PCU) e do modelo de processo *Personal Software Process* (PSP). O processo de sistematização desses dois conceitos evoluiu para a construção de uma estratégia que apóia o gerente na elaboração de um plano de desenvolvimento, na distribuição das tarefas, na estimativa de tempo e esforço e, principalmente, no acompanhamento do desenvolvimento, podendo identificar problemas e tomar medidas corretivas de forma bastante rápida (SANCHEZ, 2008).

No próximo capítulo será apresentada uma revisão sobre outro aspecto da Qualidade de Software que trata das atividades de Verificação, Validação e Testes de Software, seus principais conceitos, técnicas e critérios.

Capítulo 3

QUALIDADE DE SOFTWARE: VERIFICAÇÃO, VALIDAÇÃO E TESTE

Neste capítulo são discutidos os conceitos relativos à importância das atividades de Garantia da Qualidade de Software, com atenção especial às atividades de Verificação, Validação e Teste (VV&T) e ao processo de teste. São apresentadas também ferramentas livres que podem ser utilizadas como suporte à implantação do processo de teste de software, permitindo controle e execução de testes.

3.1 Considerações Iniciais

O processo de desenvolvimento de software é composto de diversas atividades que buscam desenvolver o software corretamente. Entretanto, mesmo que essas atividades sejam realizadas, não é possível garantir que o produto final seja liberado sem erros. A qualidade de software é um aspecto que deve ser tratado simultaneamente ao processo de desenvolvimento visto que ela não pode ser imposta depois que o produto está finalizado (MALDONADO; FABBRI, 2001a).

Segundo Pressman (2006), qualidade de software significa estar em conformidade com os requisitos do sistema, de forma que suas características implícitas e explicitas sejam atendidas. Dessa forma, a falta de conformidade com os requisitos, o não atendimento das normas estabelecidas para desenvolvimento, bem como a falta de atendimento aos requisitos não explícitos, como facilidade de uso e boa manutenibilidade, implicam em um produto de qualidade duvidável.

Segundo Lewis e Veerapillai (2004) uma das definições de qualidade, baseada no cliente, é se o produto ou serviço desempenha o que o cliente necessita. Essa adequação às necessidades se dá através do cumprimento da descrição da proposta do produto ou serviço, ou seja, os requisitos. Os requisitos compõem um dos documentos mais importantes de um projeto, sendo que o sistema de qualidade gira em torno dele e o planejamento do projeto e dos testes deve procurar atendê-lo.

Dentre as atividades de garantia da qualidade de software estão as de Verificação, Validação e Teste, ou simplesmente VV&T. Tais atividades estão, do mesmo modo que as atividades de planejamento e gerenciamento, presentes nos modelos de qualidade mais utilizados atualmente. A verificação assegura que o software ou uma determinada função sejam implementados corretamente, enquanto que a validação assegura que o software que está sendo desenvolvido está de acordo com os requisitos estabelecidos (PRESSMAN, 2006).

Andersson e Runeson (2002) em uma pesquisa empírica realizada em empresas de desenvolvimento de software identificaram que todas as empresas analisadas possuíam problemas e dificuldades em aplicar atividades de Verificação e Validação.

Em outro estudo realizado por Neto et al. (2006), foi avaliado como algumas práticas de software estão sendo utilizadas em cenários de desenvolvimento de software e foi observado que as organizações estão aplicando práticas de teste em seus projetos mas com certas limitações. Atividades como planejamento, controle, medição e análise dos testes realizados, bem como ferramentas de apoio aos testes, relacionadas à gerência ou automação dos testes, não estão sendo aplicadas nessas organizações.

A dificuldade da realização de planejamento de testes está fortemente ligada à falta de uma base de dados histórica de testes, à carência de métricas disponíveis para realizar estimativas de tempo, esforço e custo. Dessa forma, é importante promover a criação de abordagens de desenvolvimento que orientem o planejamento da atividade de teste, a criação de casos de teste, o estabelecimento e coleta de métricas de teste.

Quanto ao ferramental técnico, como softwares que auxiliam no planejamento e execução dos testes, muitas empresas ainda estão realizando poucas atividades

de teste devido ao desconhecimento das ferramentas existentes ou por possuir poucos recursos financeiros para aquisição dessas ferramentas.

A verificação e validação de sistemas usam uma parcela substancial de tempo dos cronogramas dos projetos e necessita de um olhar mais apurado (ANDERSSON;RUNESON, 2002). Dessa forma, neste capítulo serão tratados alguns conceitos relacionados a testes de software que auxiliam a entender como a atividade de testes pode ser conduzida de maneira a ser menos custosa. Na Seção 3.2 serão discutidos os principais conceitos relacionados às atividades de Verificação, Validação e Teste. Na Seção 3.3 serão apresentadas algumas ferramentas que podem ser utilizadas para auxiliar a aplicação das atividades de teste apresentadas na Seção 3.3. Por fim, na Seção 3.4 são apresentadas as considerações finais sobre os conceitos discutidos.

3.2 Verificação, Validação e Teste

As principais atividades de verificação e de validação são as inspeções e os testes que devem ser aplicadas com o objetivo de verificar a presença de defeitos no software. As inspeções consistem de atividades de análise estática, não envolvendo a execução do produto, e os testes constituem atividades de análise dinâmica, envolvendo a execução do produto de software (MALDONADO; FABBRI, 2001a). O teste de software é uma das atividades mais utilizadas tornando-se um dos elementos para fornecer evidências da confiabilidade do software em complemento a outras atividades, como por exemplo, o uso de revisões e de técnicas formais e rigorosas de especificação e de verificação (MALDONADO, 1991).

Teste de software é uma atividade crucial no processo de desenvolvimento de software (VICENZI *et al*., 2005). Segundo Black (2002), para ter uma atividade de teste de qualidade, é necessário selecionar adequadamente os testes a serem realizados, pois é fácil se perder na grande quantidade de testes existentes para garantir a qualidade de um sistema. Dessa forma, por ser uma atividade cara, é necessário ter meios de definir os testes que devem ser realizados de forma que a qualidade do software entregue seja satisfatória tanto para desenvolvedores quanto para clientes e usuários.

A atividade de teste é composta basicamente por quatro etapas: o planejamento, o projeto dos casos de teste, a execução e a avaliação dos resultados (MALDONADO; FABBRI, 2001b; TIAN, 2005). Na etapa de planejamento são estabelecidos os objetivos a serem alcançados com a atividade de teste bem como a estratégia de teste que será adotada. Na etapa de projeto de casos de testes, ou preparação, são elaborados o procedimento geral de testes e os casos de testes específicos. Nesse contexto, um caso de teste consiste da especificação do dado de teste (dado de entrada) e do comportamento esperado do programa sob teste (MALDONADO, 1991). Após a etapa de preparação, a execução dos testes é realizada e atividades como observações e medições relacionadas ao produto testado também podem ser realizadas. Por fim, a análise dos resultados da execução dos casos de testes é realizada determinando se defeitos foram encontrados, e caso tenham sido, devem ser tratados e removidos.

A organização dessas atividades pode ser descrita pelo processo genérico de testes elaborado por Tian (2005) e mostrado na Figura 3.1. Esse processo é iniciado na etapa de Planejamento e Preparação na qual, com base nas entradas estabelecidas para o processo, tais como dados, recursos e ferramentas necessárias à execução, são definidos os casos de testes e os procedimentos. Nessa etapa também são definidos os critérios para término do processo, ou seja, os objetivos de confiabilidade e cobertura necessários para definir a parada da atividade de teste e as medidas e modelos selecionados que deverão ser coletados durante a execução dos testes.

Ainda nesse processo, durante a etapa de execução dos testes, além da execução dos casos de teste são realizadas atividades como observações e medições relacionadas ao produto testado. Ao fim da execução de cada caso de teste os objetivos de confiabilidade e cobertura são verificados identificando se foram satisfeitos. Segundo Maldonado (1991), as atividades de teste não devem ser encerradas se existe insatisfação quanto à qualidade do software ou, se existe uma chance razoável de se detectar a presença de erros ainda não detectados. Para medir essa qualidade podem ser utilizadas métricas de cobertura de teste que são estabelecidas a partir do conhecimento de detalhes de implementação, com o objetivo de garantir que os testes “cobriram” todo o código (MALDONADO, 1991).

Caso os objetivos de confiabilidade e cobertura tenham sido satisfeitos, serão disponibilizados os dados ou os produtos gerados pela execução dos testes e caso

contrário, deverão ser realizadas ações apropriadas para tratar os defeitos encontrados. Complementarmente, devem ser realizadas análises sobre os resultados e medidas encontrados de maneira a prover uma realimentação apropriada ao processo de testes e ao processo de desenvolvimento em geral.

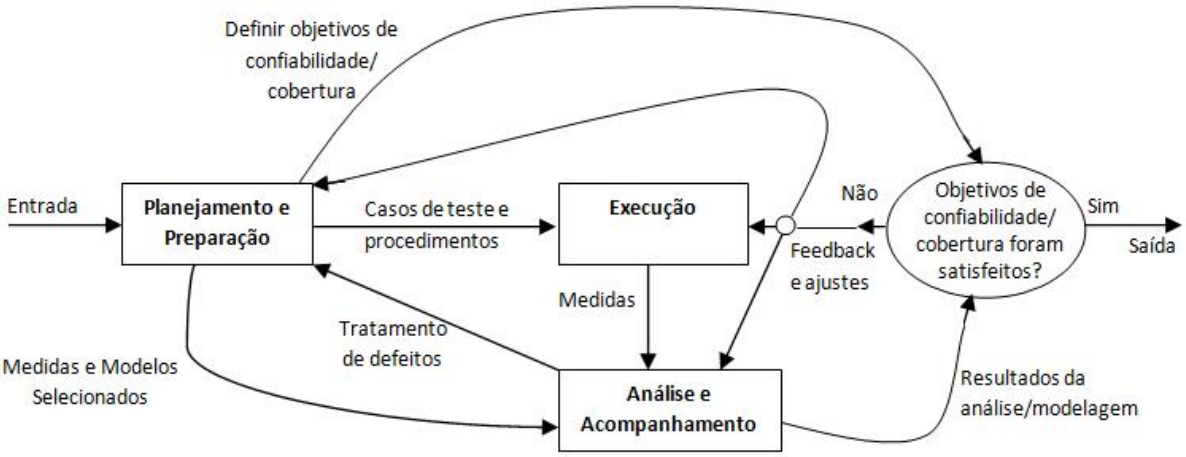


Figura 3.1 - Atividades genéricas de testes (adaptado de (TIAN, 2005))

A atividade de teste de software é dividida nas seguintes fases: teste de unidade, de integração, de validação e de sistema. O teste de unidade tem por objetivo explorar a menor unidade do sistema e identificar defeitos em cada componente ou módulo separadamente. O teste de integração, por sua vez, busca encontrar erros associados às interfaces entre os módulos quando esses são integrados. O teste de validação valida o software que acabou de ser desenvolvido de acordo com os requisitos estabelecidos do software. Já no teste de sistema o software e os outros elementos do sistema são testados como um todo (PRESSMAN, 2006).

Estruturar as atividades de teste de maneira que em cada uma das fases diferentes tipos de erros e aspectos do software sejam abordados, culmina no estabelecimento de estratégias de teste que selecionam adequadamente os dados de testes e as medidas de cobertura (MALDONADO; FABBRI, 2001b). Uma estratégia de teste de software deve integrar métodos de projeto de casos de teste em uma série bem planejada de passos, que resultam na construção bem sucedida de software. Além disso, a estratégia de teste deve ser também suficientemente flexível para promover uma abordagem de teste sob medida (PRESSMAN, 2006).

Durante a realização das atividades de teste passando pelas fases citadas anteriormente, a atividade de seleção dos dados de testes e a decisão de quando parar de testar são tomadas baseando-se em técnicas e critérios de teste.

As técnicas de teste classificam-se com base na origem das informações utilizadas para estabelecer os requisitos de teste. Assim, elas são classificadas em: técnica funcional (ou caixa preta), técnica estrutural (ou caixa branca), com base em erros e com base em máquinas de estado finito.

Os Critérios de teste servem para selecionar e avaliar casos de teste de maneira que a possibilidade de encontrar defeitos seja aumentada ou, quando isso não ocorre, estabelecer um nível elevado de confiança na correção do produto (MALDONADO; FABBRI, 2001b). Os critérios de teste são estabelecidos a partir das técnicas funcional, estrutural e baseada em erros citadas anteriormente.

A técnica funcional, também conhecida como caixa-preta ou comportamental, parte de um ponto de vista macroscópico, examinando algum aspecto fundamental do sistema dando pouca importância a estrutura lógica interna do software. O estabelecimento de seus requisitos é feito com base na especificação do software (PRESSMAN, 2006). Foca-se no comportamento externo de um sistema de software ou seus vários componentes, dessa forma a visão do objeto a ser testado funciona como uma caixa preta, prevenindo a visualização do seu conteúdo interno (TIAN, 2005).

Na técnica funcional os critérios de teste são estabelecidos a partir da especificação do software, e resumem-se a: critério por particionamento de equivalência, análise do valor limite e grafo de causa e efeito.

A técnica estrutural, também conhecida como caixa-branca ou caixa de vidro, é uma filosofia de projeto de casos de teste que usa a estrutura de controle descrita como parte do projeto ao nível de componentes para derivar casos de teste (PRESSMAN, 2006). Os requisitos de teste dessa técnica são estabelecidos com base em uma determinada implementação, verificando se os detalhes do código foram atendidos e solicitando a execução de partes ou de componentes elementares do programa (MALDONADO; FABBRI, 2001b; PRESSMAN, 2006). Por focar-se na implementação a visão do objeto a ser testado funciona como uma caixa branca permitindo que se veja seu conteúdo interno (TIAN, 2005). Os testadores examinam a estrutura interna do programa ou sistema extraindo os dados de teste sem o auxilio dos requisitos do sistema, ou seja, a sua especificação. Os critérios de teste

relacionados a essa técnica são: com base na complexidade, com base em fluxo de controle e com base em fluxo de dados.

Os critérios baseados em fluxo de controle utilizam elementos como comandos ou desvios para determinar quais estruturas devem ser testadas. Os critérios mais conhecidos baseados em fluxo de controle são o Todos-Nós, Todos-Arcos e Todos-Caminhos. O Todos-Nós ou Todos-Comandos, requer que a execução do programa passe pelo menos uma vez em cada comando do programa ou seja, em cada vértice do grafo de fluxo de controle. O Todos-Arcos ou Todos-Ramos determina que cada aresta do grafo de fluxo de controle seja exercitado pelo menos uma vez. Já o Todos-Caminhos exige que todos os caminhos possíveis de execução no programa sejam executados.

Os critérios baseados em fluxo de dados determinam os requisitos de teste baseado nas informações do fluxo de dados do programa, explorando as interações onde existem definições de variáveis e referências a essas definições. Alguns dos critérios mais conhecidos são o Def-Uso, onde são analisadas as definições das variáveis e seu uso e o Potencial-Uso onde são analisadas as definições das variáveis e os potenciais caminhos que podem fazer com que essa variável seja utilizada.

Os critérios baseados na complexidade derivam os requisitos de teste baseado na complexidade do sistema. Como exemplo tem-se o critério de McCabe que deriva os requisitos baseando-se na complexidade ciclomática do programa.

A técnica de testes baseada em erros estabelece os requisitos de teste explorando os erros mais freqüentes cometidos durante o desenvolvimento de software (DEMILLO, 1980 apud MALDONADO; FABBRI, 2001b). A ênfase da técnica está nos erros que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência (BARBOSA *et al.*, 2000). Os critérios de teste relacionados a essa técnica baseiam-se no conhecimento sobre erros típicos cometidos durante o processo de desenvolvimento e são eles a semeadura de erros e a análise de mutantes.

A técnica com base em máquinas de estado finito utiliza as máquinas de estado finito como forma de modelar o aspecto comportamental de sistemas de software utilizando a sua estrutura e o conhecimento subjacente para determinar os requisitos de teste (CHOW, 1978; FUJIWARA, 1991 apud MALDONADO; FABBRI, 2001b). Os critérios de teste baseados em máquinas de estados finito são aplicados

no contexto de validação e teste de sistemas reativos e de sistemas orientados a objetos.

3.3 Ferramentas de apoio à atividade de teste

A atividade de teste, apresentada na seção anterior, por envolver uma série de etapas e tarefas tende a ter um custo bastante elevado. Além de naturalmente custosa, se aplicada de forma incorreta ou contando somente com recursos manuais, ela pode ser bastante propensa a erros e, em alguns caso, improdutiva. O uso de ferramentas que possam apoiar a aplicação dos critérios de teste e facilitar a automatização de suas atividades propicia que um nível maior de qualidade possa ser garantido e que uma maior produtividade possa ser alcançada.

Com o objetivo de apoiar a aplicação das atividades de teste existem no mercado diversas ferramentas que dão suporte desde a etapa de planejamento até o projeto e execução de testes, gerando resultados que auxiliam a verificar a qualidade dessa atividade.

A grande maioria das ferramentas encontradas no mercado busca apoiar ou automatizar o gerenciamento de todo o ciclo de vida de uma aplicação. Conhecidas como *Application Life Cycle Management (*ALM), tais ferramentas provêem apoio ao processo completo de desenvolvimento de software. Elas visam integrar as etapas de desenvolvimento, fornecendo módulos que dão suporte ao planejamento, análise, gerenciamento de requisitos, projeto de software, testes, gerenciamento de defeitos, etc. Na área de teste de software algumas dessas ferramentas tem buscado apoiar principalmente o gerenciamento e automação do processo de teste, tornando a atividade de teste integrada com o restante do desenvolvimento.

Grandes empresas como HP e IBM alimentam o mercado com um conjunto de ferramentas que provê suporte ao ciclo de vida do desenvolvimento, com soluções relacionadas à segurança de aplicações, garantia da qualidade, arquitetura orientada a serviços, etc. Entretanto, por possuírem um alto custo se tornam inacessíveis a empresas menores, de pequeno e médio porte. Em contrapartida, existem algumas iniciativas *open source* que buscam suprir o espaço deixado por

ferramentas pagas, possibilitando a aplicação das atividades relacionadas ao projeto e desenvolvimento de software, mesmo que os orçamentos sejam reduzidos.

Na área de qualidade de software, mais especificamente na área de teste, existem iniciativas *open source* que apóiam as diversas fases da atividade de teste, principalmente no que diz respeito ás áreas de planejamento e gerenciamento de teste, automatização de testes de desempenho, funcionais e de integração. Além dessas áreas, existe também a preocupação com a geração de dados de teste, teste de segurança, teste unitário, teste de usabilidade, etc.

As ferramentas de gerenciamento de teste e de registro e acompanhamento de defeitos possibilitam que seja realizado um gerenciamento mínimo do planejamento de teste e do projeto de casos de teste. Em alguns casos as ferramentas dão suporte à execução dos testes, resultando no registro dos resultados e também na detecção de defeitos.

Já as ferramentas de automatização de testes possibilitam que os *scripts* de teste criados possam ser reproduzidos outras vezes e, dessa forma, menos tempo seja gasto na execução da atividade de teste, principalmente quando são necessários testes de regressão.

Na Tabela 3.1 apresentam-se algumas ferramentas *open source* encontradas na literatura, que foram desenvolvidas para dar suporte à aplicação das atividades de teste.

Tabela 3.1 – Ferramentas open source de suporte à atividade de testes

TIPO DE FERRAMENTA	FERRAMENTAS
Gerenciamento de testes	TestLink, rth, Testopia, TestMaster, Testitool, Test Case Web, qaManager, FireScrum, etc.
Gerenciamento de defeitos	Mantis, Bugzilla, Scarab, BugNET, Trac, etc.
Testes de desempenho	JMeter, Badboy, WebLOAD, etc.
Testes funcionais para aplicações Web	Selenium, Watir, Canoo WebTest, actiWATE, Apodora, FitNesse, etc.
Testes funcionais para aplicações Desktop	Marathon, Abbot, etc.
Cobertura de testes	Emma, Cobertura, Code Coverage, Eclemma, etc.

Tais ferramentas, se combinadas de acordo com as necessidades de cada empresa, conseguem prover suporte às atividades do processo de teste, garantindo qualidade ao ciclo de desenvolvimento de sistemas e permitindo que algumas técnicas e critérios de teste possam ser utilizados de forma fácil e sem custos.

3.3.1 TestLink e Mantis

Um exemplo bastante utilizado no cenário brasileiro por empresas de pequeno porte é a combinação das ferramentas TestLink (TESTLINK, 2010) e Mantis (MANTIS, 2010), ou mesmo, TestLink e Bugzilla, como forma de auxiliar na melhoria do processo de teste e melhoria da qualidade do produto desenvolvido.

TestLink é uma ferramenta web para gerenciamento de testes, desenvolvida em PHP e distribuída sob a licença GPL[2]. Ela pode ser integrada com outras ferramentas de gerenciamento de defeitos como Bugzilla e Mantis.

Por possuir código aberto é possível customizar a ferramenta tornando-a adequada ao uso no ambiente de desenvolvimento de cada empresa. Besson, Beder e Chaim (2009) apresentaram os resultados da criação de uma ferramenta construída com base na TestLink, a AcceptLink, que tem o propósito de ser utilizada no contexto ágil para auxiliar na modelagem e execução de casos de teste de aceitação em aplicações web.

TestLink permite que sejam criados planos de teste, projeto dos casos de teste, e que o resultado da execução dos testes (falhas, sucessos e impedimentos) possa ser também registrado. Além disso, ela permite a extração de algumas métricas relacionadas à execução dos testes e a geração de relatórios (TESTLINK, 2010).

TestLink possui as vantagens de ser escalavel, podendo ter tantos usuários quanto o hardware utilizado comportar, e também de permitir o rastreamento entre requisitos e casos de teste, possibilitando identificar se os requisitos do sistema previamente estabelecidos têm casos de teste associados. Ela fornece, em seu conjunto de funcionalidades, a possibilidade de criar estruturas que permitam a criação dos casos de teste, de projetos de teste, ciclos de teste, suítes de teste, planos de teste, e *builds* de execução. Na Figura 3.2 apresenta-se a tela de criação de um caso de teste em que devem ser inseridos o título do caso de teste (Figura 3.2 – A), uma breve descrição do caso de teste (Figura 3.2 – B), passos para execução do teste (Figura 3.2 – C) e os resultados esperados (Figura 3.2 – D).

[2] Licença para software livre que permite que os programas sob essa licença possam ser distribuídos e reaproveitados. Maiores informações podem ser encontradas no endereço: http://www.gnu.org/licenses/gpl.html

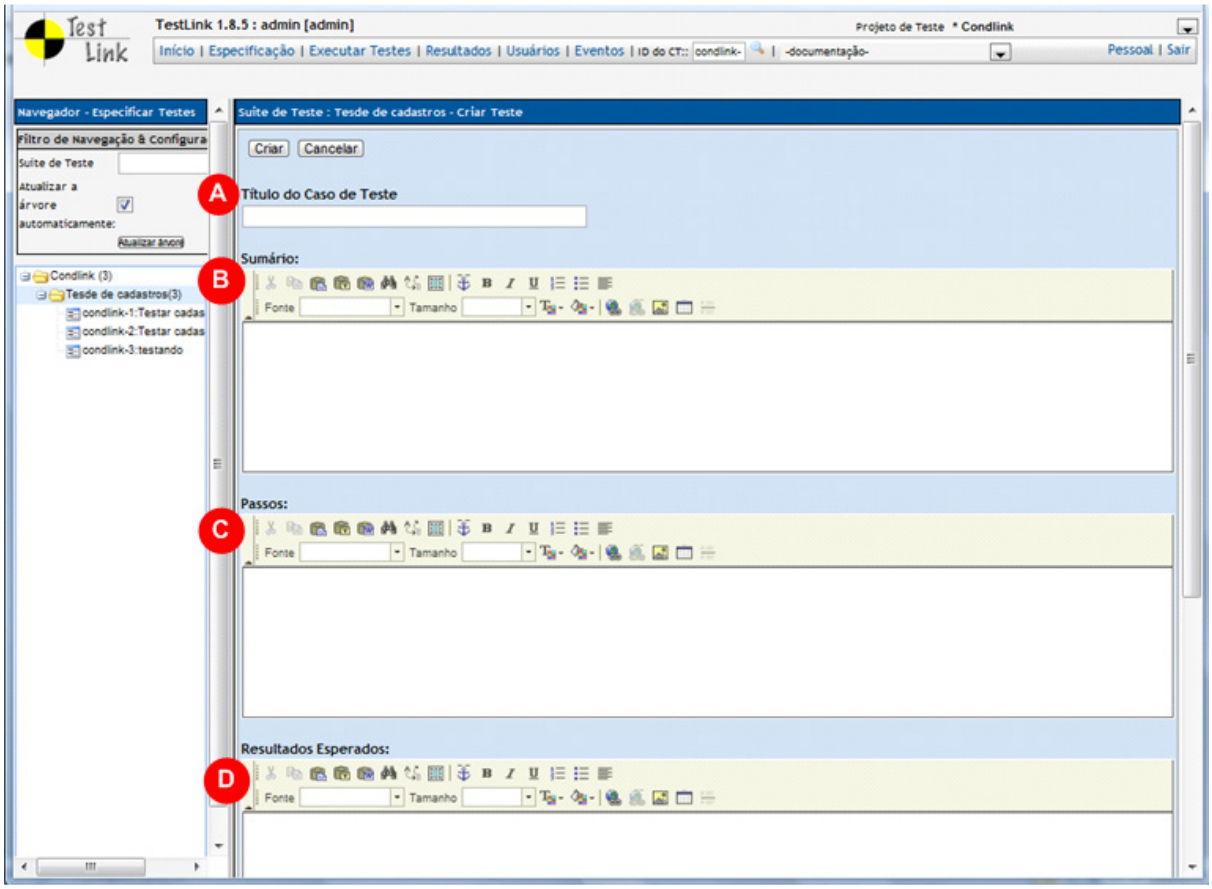


Figura 3.2 – TestLink: cadastro de um caso de teste

Mantis (MANTIS, 2010) é uma ferramenta de *bugtracking* que auxilia no cadastro, atribuição e acompanhamento de defeitos e requisições de mudança, bem como de diversas tarefas de um projeto. Assim como TestLink, Mantis também é uma ferramenta web, desenvolvida em PHP e distribuída sob a licença GPL.

Utilizando Mantis é possível tornar visíveis as atividades de teste de software durante o processo de desenvolvimento e acompanhar a resolução dos defeitos e requisições de mudança do sistema. A equipe pode acompanhar a resolução dos problemas e a realização das tarefas pela observação do status das tarefas ou podem ser notificados de alterações nos projetos por meio de *e-mail*.

Quando um novo defeito é encontrado e uma requisição de mudança é realizada, deve ser feito o registro no sistema e o status dessa solicitação deve ser selecionado. Como pode ser visualizado na base inferior da Figura 3.3, o status dos defeitos ou solicitações de mudança classificam-se em:

- Novo: atribuído à solicitação que acabou de ser criada;

- Retorno: a pessoa para a qual a solicitação foi atribuída provê algum retorno à pessoa que criou a solicitação;
- Admitido: a solicitação foi criada, mas não foi atribuída a ninguém;
- Confirmado: a pessoa para a qual a solicitação foi atribuída já recebeu e confirmou o recebimento;
- Atribuído: solicitação já foi atribuída a alguém;
- Resolvido: solicitação resolvida;
- Fechado: após a solicitação ser resolvida deve ser conferida, ou mesmo re-testada, por quem atribuiu a solicitação.

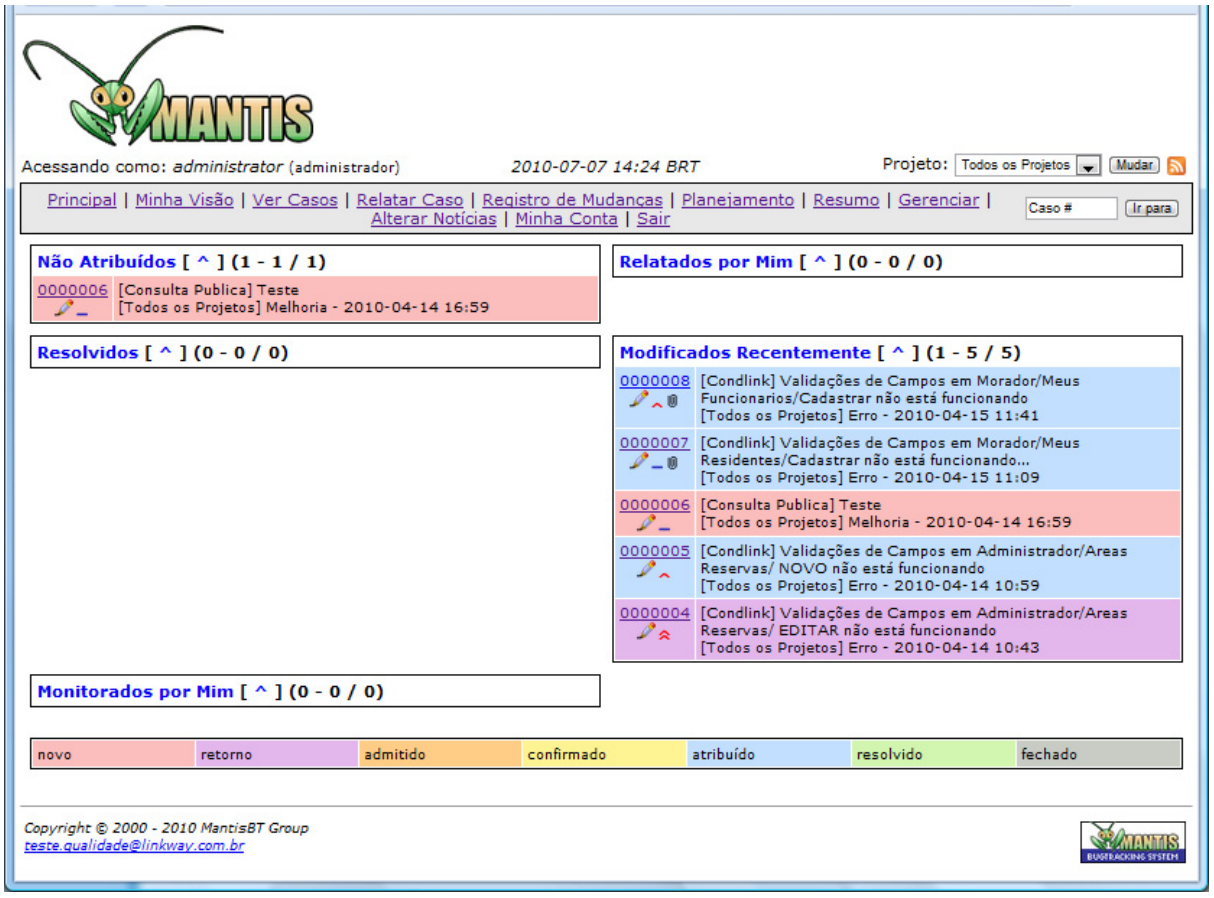


Figura 3.3 – Tela principal do Mantis onde são exibidos as solicitações realizadas classificados por status

Em uma situação em que as ferramentas TestLink e Mantis estejam sendo usadas de forma integrada, é possível associar o defeito ao caso de teste que detectou esse defeito. Durante a execução dos testes especificados na ferramenta TestLink, caso algum teste provoque uma falha, quando o defeito for encontrado

pode ser reportado diretamente nessa mesma ferramenta e associado a uma nova solicitação na ferramenta Mantis. Para tanto é necessário alterar o status do caso de teste na TestLink para “Com Falha” (Figura 3.4 – A) e selecionar o link que será exibido com ícone em forma de “bug” no campo Gerenciamento de Casos para reportar o problema (Figura 3.4 – B). Nessa situação, na qual as ferramentas estão configuradas para atuarem em conjunto, será aberta uma tela que permite acessar o Mantis (Figura 3.4 – C) e realizar o registro do defeito.

Em um ambiente em que não houvesse essa integração, o Mantis deveria ser aberto e o defeito deveria ser registrado, mas o rastreamento de qual caso de teste possibilitou que o defeito fosse revelado estaria perdido.

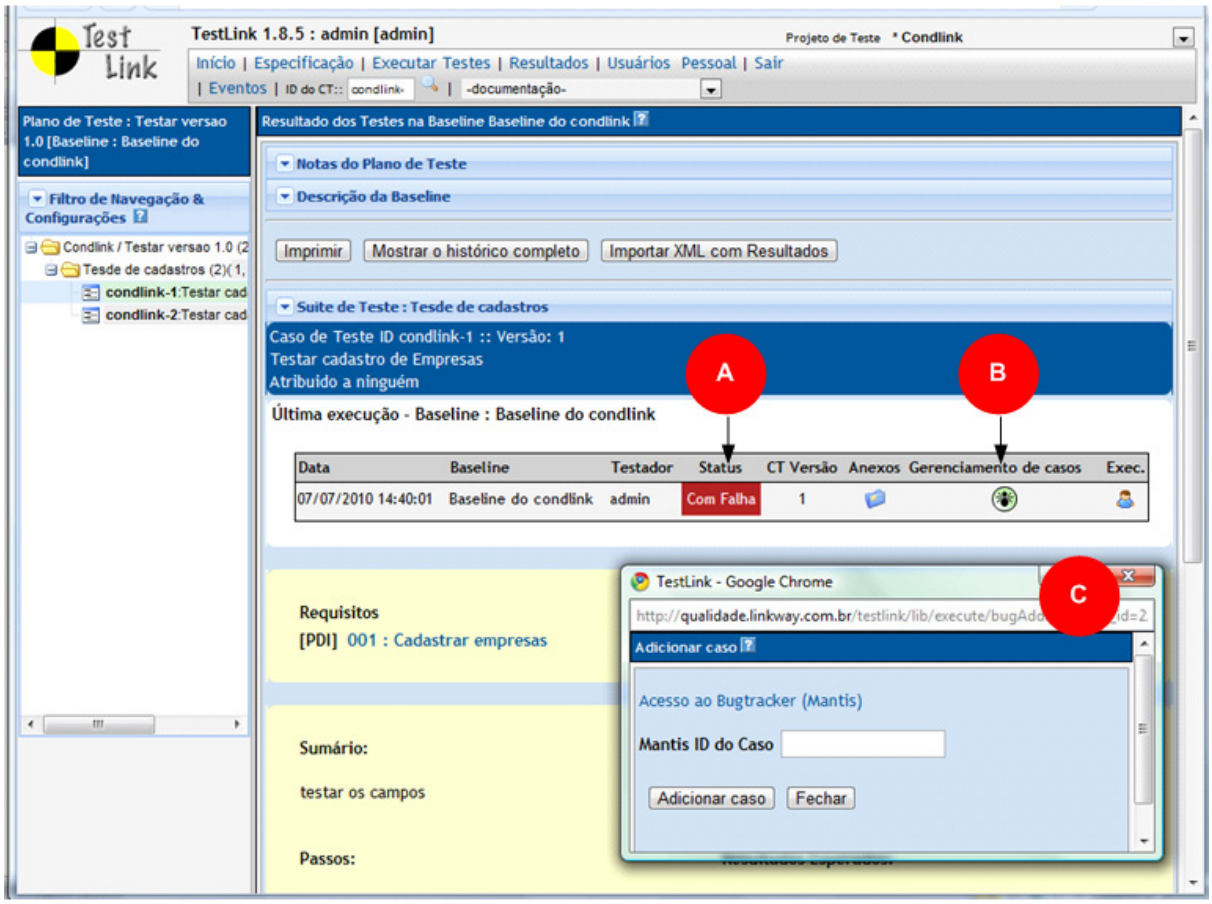


Figura 3.4 – Integração entre Mantis e TestLink

A integração entre TestLink e Mantis permite a implantação de um ambiente que provê um determinado controle da atividade de teste, propiciando uma maior visibilidade dessas atividades a todos os envolvidos no desenvolvimento do software. Além disso, como mencionado anteriormente, apenas com o uso dessas

ferramentas já se torna possível estabelecer rastreabilidade entre requisitos, testes e defeitos encontrados, possibilitando a derivação de medidas sobre a qualidade da atividade de teste realizada.

3.3.2 Selenium

A execução automatizada de teste difere da execução manual pelo fato dos casos de teste serem criados e executados utilizando alguma ferramenta apropriada que permita que eles sejam repetidos automaticamente. A execução automatizada se justifica em situações em que as atividades sejam repetitivas ou difíceis de serem realizadas manualmente, e que o ganho em tempo justifique o esforço de automatização. Entretanto, caso o sistema sob teste seja simples, pode não ser vantajoso automatizar e sim priorizar a execução manual.

Dentre as ferramentas existentes para automatização de testes funcionais web mostradas na Tabela 3.1 a Selenium (SELENIUM, 2010) mostra-se como uma das mais utilizadas, de acordo com pesquisas realizadas em fóruns e grupos de discussão sobre automatização de testes para aplicações *web*. A Selenium possui um conjunto de ferramentas *open source* que permite a criação de scripts simples de teste que são executados com o uso de um browser.

Os testes são criados baseados numa série de comandos denominados *selenese* que permitem que determinadas ações sejam realizadas. Uma sequência de comandos é denominada *script* de teste e o agrupamento desses *scripts* resulta em uma suíte de testes. Os comandos existentes para execução dos testes permitem abrir uma página web, inserir valores em campos, selecionar opções, clicar em links e botões, realizar verificações, dentre outras ações. Esses comandos são agrupados em três grupos denominados *Actions*, *Acessors* e *Assertions*. Os comandos do tipo *Action* permitem que as ações realizadas provoquem alguma mudança de estado na aplicação, como a abertura de uma página, o clique em um botão, etc. Os comandos *Acessors* analisam o estado da aplicação sob teste armazenando o resultado em uma determinada variável. Já os comandos *Assertion* verificam se o estado da aplicação está conforme o esperado, permitindo, por exemplo, verificar se um determinado texto está presente em uma página web.

As ferramentas que compõem a família Selenium são o Selenium-IDE, Selenium-RC (*Remote Control*) e Selenium-Grid. O Selenium-IDE é um ambiente de

desenvolvimento integrado de casos de teste que funciona como um plug-in do Firefox possibilitando a gravação de *scripts* e depois a sua execução. Ele captura as ações realizadas na aplicação sob teste, grava os passos executados e informações trocadas com a aplicação e gera um *script* contendo esses passos. Esse *script* pode então ser reproduzido e as ações gravadas podem ser novamente executadas. As linguagens em que os *scripts* podem ser criados são: HTML, Java, C#, Perl, PHP, Python, e Ruby.

O Selenium-RC permite que *scripts* de teste com estruturas mais complexas possam ser criadas por meio da utilização de uma linguagem de programação. Uma API (*Application Programming Interface*) é fornecida para que o testador possa criar casos de teste que contenham, por exemplo, uma iteração.

Já o Selenium-Grid possibilita que várias instâncias do Selenium-RC sejam executadas em vários sistemas operacionais e com diferentes configurações de hardware ao mesmo tempo. Dessa forma, é possível escalonar grandes suítes de teste reduzindo o tempo de execução e também permitir a execução de suítes em diferentes ambientes.

Holmes e Kellog (2006) apresentam os resultados e aprendizados da aplicação da Selenium em um ambiente ágil de desenvolvimento. Após experimentarem outras ferramentas como Canoo Web Test e HttpUnit, optaram por utilizar a Selenium. Dentre as vantagens apresentadas, eles destacaram a capacidade de reproduzir cada ação do usuário realizada na aplicação web, a possibilidade de escrever extensões próprias e de adicionar ações customizadas de forma a sofisticar mais a ferramenta.

3.3.3 Cobertura

Cobertura é uma ferramenta livre de cobertura de código para sistemas desenvolvidos em Java. Ela calcula a porcentagem de código que é acessado durante a execução dos testes (COBERTURA, 2010).

Quando os testes são executados, seja de forma manual ou automatizada, é importante verificar se estão sendo cobertos pontos importantes do sistema determinados pelos critérios adotados na ferramenta que são o Todos-Nós ou Todos-Ramos. Com a aplicação de uma ferramenta de cobertura, é possível obter os resultados da execução dos testes e adequar os dados de teste passados como

entrada de forma a aumentar a cobertura ou cumprir os critérios existentes na ferramenta. Assim, considerando a ferramenta Cobertura, como ela adota os critérios Todos-Nós e Todos-Ramos, caso o planejamento dos testes tenha sido realizado com base em um destes critérios é possível verificar por meio dela qual a cobertura obtida com a execução dos testes até o momento.

A cobertura é obtida por meio da instrumentação do *bytecode* do sistema, ou seja, diretamente das classes Java compiladas. É possível coletar os dados de execução de três formas: por meio da inserção de instrumentação no código fonte, por meio da adição de instrumentação no *bytecode* e também pela utilização de uma máquina virtual Java que rode o código a ser analisado.

Quando as instruções de instrumentação inseridas no código são encontradas pela máquina virtual Java, durante a execução do sistema ou da aplicação dos testes, o código inserido pela ferramenta de cobertura incrementa diversos contadores no código. Dessa forma é possível dizer quais instruções foram executadas e quais não foram.

Uma classe instrumentada serializa as instruções em um arquivo denominado *cobertura.ser* que posteriormente será lido e, a partir das informações armazenadas, poderão ser gerados relatórios de execução no formato HTML ou XML.

Nas Figura 3.5 e Figura 3.6 são apresentados relatórios gerados pela Cobertura. No relatório da Figura 3.5 são exibidas as informações gerais de execução, sendo que para cada pacote são exibidas a quantidade de classes, a porcentagem de linhas cobertas, a porcentagem de ramos cobertos e a complexidade ciclomática.

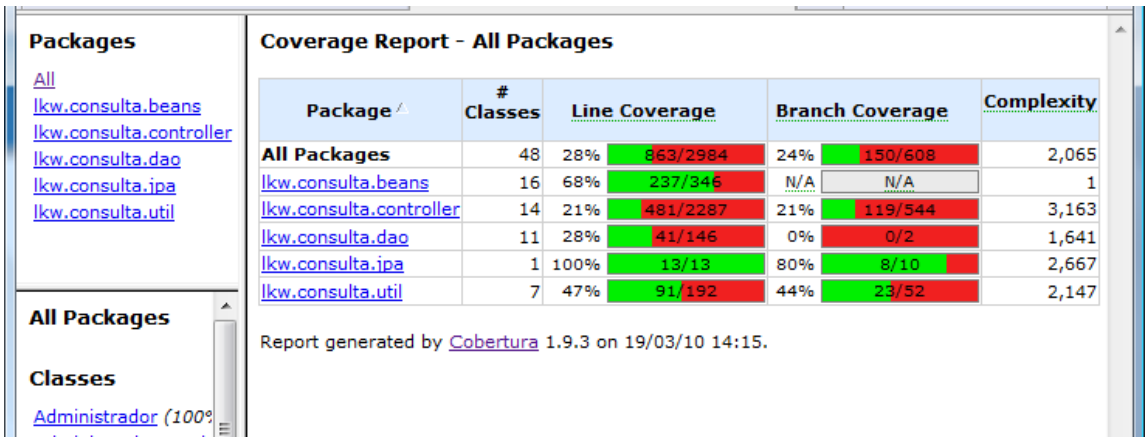


Figura 3.5 – Relatório geral de cobertura da ferramenta Cobertura

Na Figura 3.6 são exibidas as informações de execução geradas com base somente em uma classe. As linhas que foram cobertas durante a execução são mostradas com sua numeração marcada na cor verde e as linhas que não foram executadas são marcadas em vermelho. Na frente de cada linha é exibida a quantidade de vezes que aquela linha foi executada.

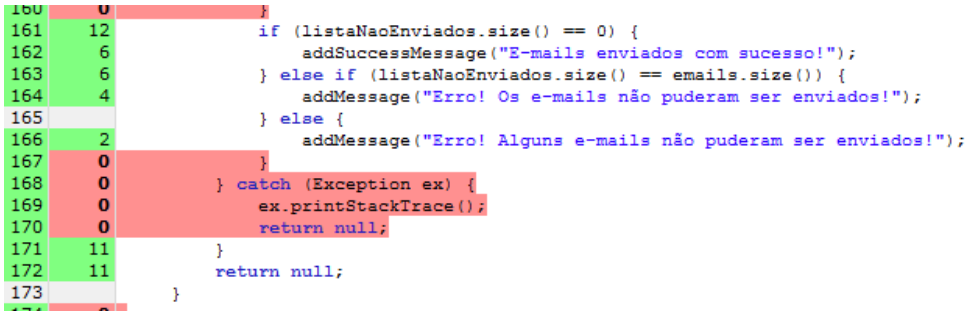


Figura 3.6 – Relatório de cobertura em uma classe usando a ferramenta Cobertura

Com base nos relatórios gerados, é possível tomar decisões sobre quais testes podem ser feitos para incrementar a cobertura total de um sistema. Detalhes sobre a utilização dessa ferramenta podem ser vistos no Anexo A.

3.4 Considerações finais

Neste capítulo foram apresentados conceitos relacionados ao processo de teste, técnicas e critérios de testes. As atividades de teste fazem parte das atividades de verificação e validação aplicadas com o intuito de garantir que o projeto de software, a codificação e a documentação estejam de acordo com os requisitos previamente estabelecidos (LEWIS; VEERAPILLAI, 2004). Assim, por meio do estabelecimento dessas atividades é possível verificar se os requisitos do sistema inicialmente estabelecidos estão sendo atendidos.

Foram também abordadas neste capítulo as ferramentas existentes que podem auxiliar na realização das atividades de teste com um maior enfoque em ferramentas *open source*. Por meio da inserção de ferramentas que facilitem que as atividades de teste possam ser realizadas a um baixo custo, é possível tornar as

atividades de garantia da qualidade acessíveis qualquer tipo de empresa, desde as de grande até as de pequeno porte.

Além das ferramentas apresentadas anteriormente que podem ser usadas de forma conjunta para prover suporte às atividades de teste, existem ferramentas que assim como os ALMs, conseguem organizar diferentes funcionalidades em uma mesma aplicação, provendo uma solução conjunta para o acompanhamento de um sistema. Dentre elas, a FireScrum, que foi utilizada neste trabalho, mostrou-se como uma boa opção, principalmente para equipes que utilizam o *framework* ágil Scrum. Esse ferramenta será comentada no próximo capítulo, que trata mais especificamente dos conceitos ágeis e da sua relação com o planejamento de projetos de software e com as atividades de garantia da qualidade apresentadas neste capítulo. Como será visto, dentre as funcionalidades existentes na FireScrum, algumas estão relacionadas ao planejamento e gerenciamento de testes e ao registro e acompanhamento de defeitos.

Capítulo 4

MÉTODOS ÁGEIS

Devido à importância e à disseminação que as metodologias ágeis vêm ganhando com o decorrer do tempo, neste capítulo serão apresentadas suas principais características, o seu ciclo de vida e os métodos mais utilizados. Serão discutidas as abordagens ágeis de gerência e planejamento de projetos bem como de VV&T, assuntos abordados nos capítulos anteriores e revisitados aqui sob a ótica ágil. Além disso, serão também mostradas ferramentas que podem auxiliar a aplicação das atividades de gerência e planejamento de desenvolvimento e teste de software a um baixo custo.

4.1 Considerações Iniciais

O processo de desenvolvimento de software passou, ao longo dos anos, por diversas transformações de forma a adaptar-se às necessidades do mercado que se tornou cada vez mais competitivo. Enquanto o prazo para entrega dos sistemas diminuiu, o tamanho das aplicações aumentou e a qualidade do software desenvolvido tornou-se cada vez mais essencial.

Como conseqüências dessas mudanças, alguns problemas surgiram em todo o ciclo de desenvolvimento de software tais como o atraso de entregas, o cancelamento de projetos, defeitos nas aplicações, não-adequação do produto às necessidades do cliente, mudança nos requisitos do negócio, e alta rotatividade de colaboradores nas empresas (MAHER, 2009). Dessa forma, ao contrário do desenvolvimento tradicional, as empresas têm que lidar mais de perto com esses problemas e com a instabilidade constante no processo de desenvolvimento.

Boehm (2002) discute que, enquanto o desenvolvimento tradicional possui em suas práticas extensos planejamentos e processos burocráticos, a nova geração de desenvolvedores procura lutar contra a burocracia dos processos engessados almejando uma maior flexibilidade. Nesse novo contexto, os problemas devem ser antecipados e uma metodologia de desenvolvimento que seja capaz de acomodar de forma flexível todos estes potenciais obstáculos deve ser adotada.

Nesse cenário, as metodologias ágeis vêm adquirindo cada dia mais importância por introduzirem uma maior flexibilidade ao processo de desenvolvimento. Adotam como princípios a adaptação a mudanças, menor importância à documentação e sim às pessoas e à entrega de software funcionando. O desenvolvimento iterativo e a participação constante do cliente possibilitam a construção de um produto de software mais adequado à necessidade do cliente e em menor tempo. O crescimento desse paradigma de desenvolvimento iniciou ainda na década de 90 com a disseminação do XP (BECK, 2004), e posteriormente do Scrum (SCHWABER, 2004), Crystal (COCKBURN, 2002), *Feature-Driven Development* (PALMER; FELSING, 2002) e outras metodologias. Hoje, tais metodologias estão estabelecidas em diversos níveis no âmbito da academia, educação e no desenvolvimento profissional de software (BEGEL, NAGAPPAN, 2007).

Dada a importância dessa nova abordagem para a área de desenvolvimento de software, neste capítulo serão apresentados os principais conceitos relacionados às metodologias ágeis, e a sua relação com as áreas de planejamento e gerenciamento de projetos e de VV&T. Desse modo, este capítulo encontra-se organizado da seguinte forma: na Seção 4.2 serão caracterizadas as metodologias ágeis, seus principais conceitos, métodos mais utilizados atualmente e o ciclo de vida de desenvolvimento ágil; na Seção 4.3 será apresentado o ciclo de vida básico das metodologias ágeis; na Seção 4.4 os conceitos relacionados ao Planejamento e Gerenciamento de projetos serão abordados no contexto ágil; na Seção 4.5 serão discutidas as questões relacionadas à Qualidade de Software, mais especificamente as atividades de VV&T, sob o ponto de vista ágil; na Seção 4.6 serão apresentadas algumas ferramentas de apoio ao planejamento e gerenciamento de projetos que podem ser utilizadas no contexto ágil; e por fim, na Seção 4.7 serão apresentadas as considerações finais sobre os tópicos abordados neste capítulo.

4.2 Caracterização das metodologias ágeis

As metodologias ágeis tiveram seu ponto de partida em fevereiro de 2001 quando, Kent Beck e outros 16 pesquisadores assinaram o Manifesto Ágil (MANIFESTO, 2001). Esse manifesto denota a filosofia do desenvolvimento ágil em 4 pontos principais:

- Indivíduos e interações são mais importantes que processos e ferramentas;
- Software funcionando é mais importante que extensa documentação;
- Colaboração do cliente tem maior valor do que negociação de contrato;
- Resposta rápida às mudanças tem maior valor que seguir os planos pré-estabelecidos.

Esses 4 valores originaram 12 princípios que sustentam o desenvolvimento ágil:

1 – A maior prioridade é satisfazer o cliente por meio da entrega contínua e rápida de software que possua valor.

2 – Mudanças nos requisitos são bem vindas, mesmo que de última hora, ou realizadas tardiamente no processo de desenvolvimento.

3 – Frequentemente deve-se entregar versões do software funcionando.

4 – Clientes, ou representantes do cliente, devem trabalhar em conjunto com desenvolvedores ao longo de todo o projeto.

5 – O projeto deve ser construído com pessoas motivadas. Para tanto, deve ser disponibilizado o ambiente e o suporte necessários, e é imprescindível confiar no potencial da equipe.

6 – A conversa “cara-a-cara” é o método mais eficiente para colher informações sobre o projeto com o cliente.

7 – Software funcionando é a melhor medida de progresso do projeto e não documentação.

8 – Processos ágeis promovem desenvolvimento sustentável. Stakeholders, clientes e desenvolvedores devem descobrir o ritmo de trabalho e mantê-lo constantemente.

9 – Atenção contínua à excelência técnica e a um bom projeto promove a agilidade.

10 – Simplicidade é essencial.

11 – As melhores arquiteturas, requisitos e projeto provêm de equipes auto-organizadas.

12 – A equipe deve refletir, em intervalos regulares, como se tornar mais eficiente e, dessa forma, deve ajustar o seu comportamento conforme as necessidades.

Essas mudanças de paradigma sobre o processo de desenvolvimento de software denotam uma das principais diferenças entre as metodologias tradicionais e as metodologias ágeis. As equipes de desenvolvimento, expostas a essas novas abordagens, devem absorver esses novos princípios e pensar no processo de desenvolvimento de forma diferente da tradicional com entregas de produto sendo realizadas de forma contínua, iterativa e incremental. Esse novo paradigma auxilia a equipe de software a focar-se nas solicitações do cliente seguindo a priorização do que deve ser feito, tendo o cliente validando o sistema continuamente.

Canós e outros (2003) levantaram algumas das principais diferenças entre as metodologias tradicionais de desenvolvimento e as metodologias ágeis. Enquanto os métodos tradicionais baseiam-se nos padrões pré-estabelecidos pelos modelos de processo, as metodologias ágeis baseiam-se em dados estatísticos obtidos por meio do levantamento histórico de implementação da equipe.

As metodologias tradicionais oferecem resistência a mudanças por várias razões, tais como o fato de se basearem em modelos de desenvolvimento, arquiteturas e processos pré-estabelecidos, o que torna o desenvolvimento do projeto dependente das diretrizes propostas por esses modelos; além disso, possuem um controle muitas vezes inflexível da documentação de seus processos e contratos; são também características peculiares dessas metodologias a limitação quanto à participação dos clientes e a grande quantidade de integrantes na equipe.

Por outro lado, as metodologias ágeis são mais flexíveis a mudanças nos requisitos do projeto; nesse caso, a forma de trabalho é definida pelos membros da equipe, o controle sobre os processos é reduzido, priorizando a conclusão do trabalho, os contratos são firmados de forma mais flexível, já que podem ocorrer mudanças no decorrer do projeto; quanto ao cliente, este participa da equipe,

influenciando e priorizando o trabalho a ser desenvolvido; as equipes são compostas por menos pessoas e os modelos utilizados representam o que a equipe considera essencial para o desenvolvimento do projeto.

Embora, em comparação aos métodos tradicionais as metodologias ágeis tenham as suas vantagens, a sua aplicação depende do contexto de desenvolvimento em que deve ser aplicado, e também do contexto para o qual o software está sendo desenvolvido. Caso o sistema a ser desenvolvido seja de grande risco e necessite de uma grande confiabilidade, um método tradicional pode ser mais adequado. Na Seção 4.2.1 seguinte serão apresentadas as principais diferenças entre os métodos ágeis existentes e as peculiaridades do ciclo de vida do desenvolvimento de software ágil.

4.2.1 Métodos Ágeis

Os Métodos Ágeis implementam os princípios das metodologias ágeis sendo que os mais populares são o *Extreme Programming* – XP (BECK, 2004), o Scrum (SCHWABER, 2004), o *Feature Driven Development* (PALMER; FELSING, 2002),o *Dynamic System Development Method* –DSDM (DSDM, 2010), o *Adaptive Software Development* – ASD (HIGHSMITH, 2002), o *Crystal Clear* (COCKBURN, 2002) dentre outros.

Abrahamsson e outros (ABRAHAMSSON et al., 2002) fizeram um comparativo entre os principais métodos com base nas suas principais características e escopo de uso. Os pontos-chave de cada método e as características especiais que diferenciam um método do outro. Esses resultados são apresentados na Tabela 4.1.

Tabela 4.1 – Principais métodos ágeis (adaptado de (ABRAHAMSSON *et al.*, 2002))

Nome do método	Pontos chave	Características especiais	Deficiências identificadas
Adaptive Software Development (ASD)	Cultura adaptativa, colaboração, desenvolvimento iterativo e incremental baseado em componentes	Desenvolvimento de sistemas complexos e de grande porte nos quais as organizações são vistas como sistemas adaptativos	ASD relaciona-se mais aos conceitos e cultura organizacional que em práticas de software
Crystal	Família de métodos que possuem os mesmos princípios centrais, entretanto variam quanto às técnicas, papéis, ferramentas e padrões	Habilidade de selecionar o método mais adequado com base no tamanho do projeto e no nível crítico	Apenas dois dos quatro métodos sugeridos estão em uso efetivo: o Crystal Clear e o Crystal Orange
Dynamic System Development Method (DSDM)	Aplicação de controles para o Rapid Application Development (RAD), uso de time boxes (período de tempo pré-estabelecido pela equipe para desenvolvimento)	De fato o primeiro método de desenvolvimento ágil. Usa a prototipação e possui diversos papéis, dentre eles: embaixador, visionário e conselheiro	Somente membros do consórcio DSDM possuem acesso a artigos com aplicações atuais do método
Extreme Programming (XP)	Desenvolvimento dirigido ao cliente, equipes pequenas, construções (*builds)* diárias	Refatoração - o redesenho permanente do sistema para melhorar o seu desempenho e capacidade de resposta à mudança	Enquanto as práticas individuais são adequadas para diferentes situações, menos atenção foi dada às práticas de gerenciamento e visão geral do sistema
Feature Driven Development (FDD)	Composto por cinco fases simples, desenvolvimento baseado em componentes orientados a objeto, iterações curtas que variam de horas a duas semanas	Simplicidade do método, projeto e implementação do sistema por features, modelagem de objetos	Está focado somente em projeto e implementação, necessitando outras abordagens que lhe dêem suporte
Scrum	Independente, pequeno, equipes de desenvolvimento auto-organizadas, ciclos de entrega de no máximo 30 dias	Impõe uma mudança de paradigma do “definido e repetível” para “nova visão de desenvolvimento de produtos” do Scrum	É detalhado como gerenciar o ciclo de entrega de 30 dias, entretanto não há nenhum detalhamento sobre a realização de testes de aceitação e integração

Dentre os métodos apresentados o ASD é o método mais abstrato do ponto de vista do desenvolvimento de software, e seus praticantes possuem grande dificuldade em traduzir os seus conceitos para o uso prático. Por outro lado, o XP representa o ponto de vista do desenvolvimento prático, contendo algumas práticas comprovadas empiricamente, como a refatoração, que se mostrou bastante útil para os seus praticantes. Os métodos da família Crystal são os únicos que sugerem explicitamente princípios de projeto baseados no tamanho e criticidade do sistema.

O DSDM se diferencia dos outros métodos por fazer uso da prototipação e pela criação de papéis que os outros métodos não consideram como, embaixador, conselheiro e visionário. Esses papéis permitem representar os diferentes pontos de vista dos consumidores do sistema. O FDD não se preocupa em solucionar todos os problemas relativos ao desenvolvimento de software e sim em focar 5 passos simples que se baseiam em identificar, projetar e implementar funcionalidades. E por fim, o Scrum é uma abordagem de gerenciamento de projetos que se baseia na auto-organização das equipes de projeto que buscam desenvolver o trabalho selecionado em ciclos de, em média, 30 dias, denominados *Sprints*.

Mesmo possuindo diversos atributos que substituem o uso dos métodos tradicionais, os métodos ágeis podem não ser adequados para todas as fases do ciclo de vida de desenvolvimento de software. Na Figura 4.1 é possível visualizar quais fases do processo de desenvolvimento são consideradas pelos métodos ágeis citados anteriormente. Para entender a figura deve-se observar que cada método possui três blocos diferentes: o primeiro representa suporte ao gerenciamento de projeto; o segundo identifica se o processo sugerido pelo método para ser aplicado é descrito juntamente com o método; e o terceiro indica se o método descreve práticas, atividades e produtos de trabalho que podem ser usados em diferentes circunstâncias. A cor cinza indica quando um bloco cobre uma fase do ciclo de vida e a cor branca indica que o método não fornece informação detalhada sobre uma das três áreas avaliadas, gerenciamento de projeto, existência de processos ou práticas/atividades/produtos de trabalho.

É possível perceber na Figura 4.1, que cada método foca em diferentes aspectos do ciclo de vida de desenvolvimento de software. Enquanto o XP está mais focado em práticas de desenvolvimento, o Scrum está mais focado no gerenciamento do projeto. Dentre os métodos citados, somente o DSDM provê cobertura total ao ciclo de vida de desenvolvimento e a maioria deles é adequada para a fase de especificação de requisitos. Com base nessas informações é possível optar por um determinado método para ser usado no desenvolvimento de um projeto. Além das informações visualizadas na Figura 4.1, a quantidade de membros que compõem a equipe pode ser outro fator decisivo para a escolha de um desses métodos. Enquanto o XP e o Scrum focam em equipes pequenas, com menos de 10 membros, os métodos Crystal, FDD, ASD e DSDM afirmam serem capazes de gerenciar até 100 desenvolvedores (ABRAHAMSSON *et al.*, 2002).

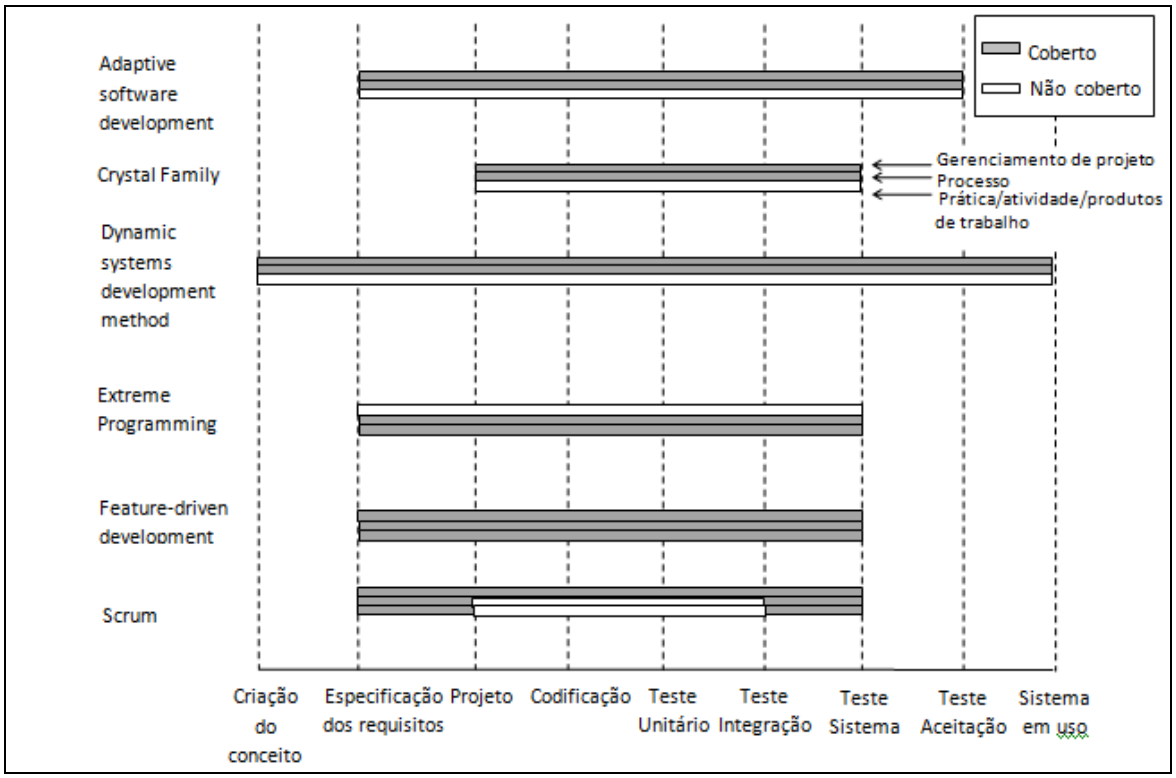


Figura 4.1 - Suporte ao ciclo de desenvolvimento de software (adaptado de (ABRAHAMSSON *et al.*, 2002))

4.2.2 Ciclo de vida nos métodos ágeis

Ambler (2010b) afirma que o escopo do ciclo de vida de desenvolvimento de software pode variar bastante pelo fato do desenvolvimento de sistemas ser complexo e por existir muito mais a ser analisado, relacionado à tecnologia da informação, que somente ao desenvolvimento. Assim, para ter sucesso no desenvolvimento de um projeto é necessário que o método cubra mais do que o desenvolvimento do software.

Ambler (2010b) sugere que o ciclo de vida possua mais fases que as apresentadas, por exemplo, no Scrum, e que cubram todo o ciclo de vida do desenvolvimento do sistema (*System Development Life Cycle* – SDLC). O SDLC ágil proposto por Ambler é composto de seis fases que são Iteração -1 (menos um), Iteração 0 (zero), Construção, Release (Fim de jogo), Produção e Aposentadoria (*Retirement*). Tais fases podem ser visualizadas na Figura 4.2.

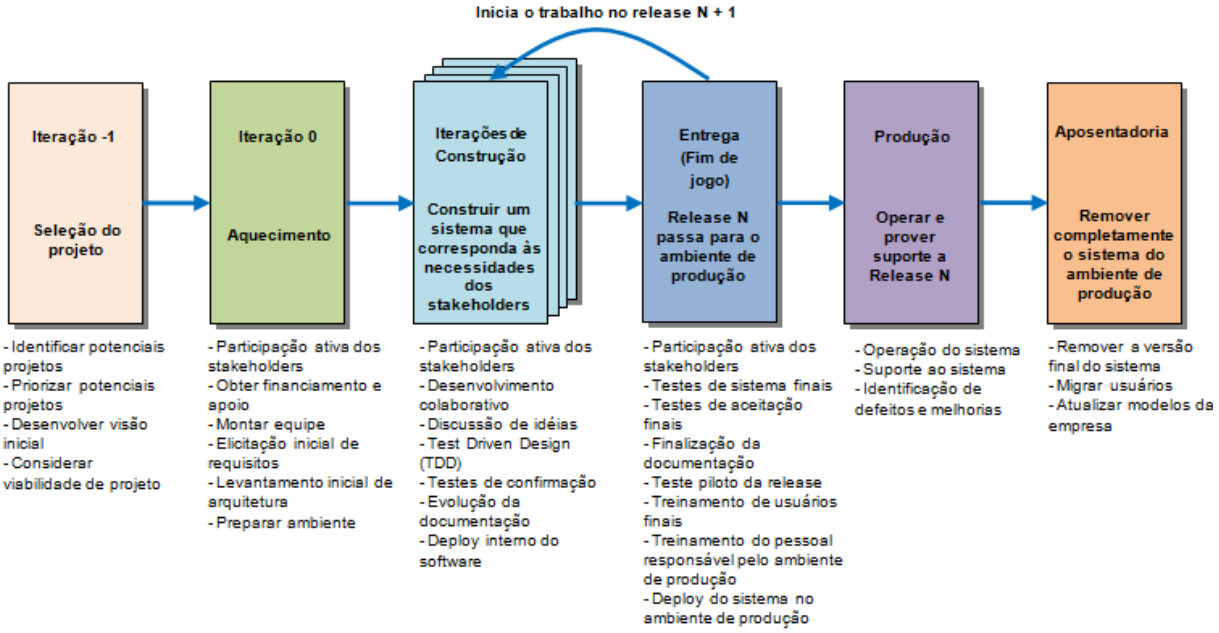


Figura 4.2 – Ciclo de vida ágil do desenvolvimento de sistema – SDLC (adaptado de (AMBLER, 2010b))

Durante a fase Iteração -1, ou planejamento pré-projeto, é realizada uma análise inicial do negócio ou projeto em questão procurando entender os objetivos e necessidades dos *stakeholders*. Por meio dessa análise inicial é possível identificar: a viabilidade de realização do projeto; a estratégia que deverá ser aplicada para o seu desenvolvimento; se será necessária alguma parceria para o desenvolvimento do sistema e outras questões importantes. Essas atividades devem ser realizadas da forma mais ágil possível, contando com a colaboração dos stakeholders para obter um entendimento do sistema que permita tomar a decisão de levar ou não adiante o projeto.

A fase Iteração 0 ou Aquecimento refere-se à primeira semana ou primeira iteração. Nessa fase o projeto é iniciado e as principais atividades referem-se a obter financiamento e suporte para o projeto, montagem da equipe de desenvolvimento, modelagem da arquitetura inicial do sistema e preparação do ambiente. Além dessas atividades, o contato constante com os clientes do projeto permite obter um escopo inicial do sistema que será refinado durante as outras iterações.

Durante as Iterações de Construção o software é desenvolvido incrementalmente e de forma colaborativa de acordo com as necessidades dos stakeholders. As funcionalidades são desenvolvidas de acordo com a prioridade elencada pelos clientes, que detém o controle sobre o escopo, orçamento e

cronograma do projeto. Nessa fase são realizadas ainda a análise e projeto do software a ser desenvolvido. O desenvolvimento será realizado seguindo o *test-driven design* (TDD) no qual após a escrita de um teste deve ser escrito o código que será validado por esse teste. As entregas devem ser feitas regularmente e a busca pela qualidade do produto desenvolvido deve ser constante. A qualidade deve ser alcançada seja por meio da padronização do código e de estilos de codificação, como também por meio da realização de testes confirmatórios que combinam testes que confrontam o código desenvolvido e testes de aceitação que confrontam os requisitos do sistema.

Na fase de Entrega ou fim de jogo (*End Game*), o sistema é transposto do ambiente de desenvolvimento para o ambiente de produção. Nesse momento são realizados testes de sistema e de aceitação finais, que não correspondem à maior parte dos testes que se espera que tenham sido realizados na fase de construção. Nessa fase ainda pode ser realizado algum retrabalho para correção dos defeitos encontrados e também a finalização da documentação do sistema e do usuário. Por fim, são realizados treinamentos para os usuários do sistema e do pessoal responsável pelo suporte ao sistema.

A próxima fase, de Produção, tem por objetivo manter o sistema desenvolvido utilizável e produtivo depois de ter sido instalado no ambiente do usuário, ou seja, manter o sistema rodando e prover suporte ao usuário. Essa fase se encerra quando o suporte dado ao sistema se encerra ou quando o software se torna obsoleto, caindo em desuso e entrando na fase de Aposentadoria ou *Retirement*. Nessa fase uma determinada versão do sistema é desativada ou o sistema inteiro é removido do ambiente de produção.

Cada método ágil apresentado na Seção 4.2.1 possui o seu próprio ciclo de desenvolvimento. Para ilustrar, na Figura 4.3 é apresentado o ciclo de desenvolvimento do Scrum em que os requisitos correspondem a uma pilha de itens priorizados que devem ser selecionados para realização na iteração. As iterações no Scrum são denominadas *Sprints* que possuem um tempo determinado em dias corridos, os quais podem ser adequados para cada empresa. Schwaber (2004) sugere que o *Sprint* ocorra em 30 dias.

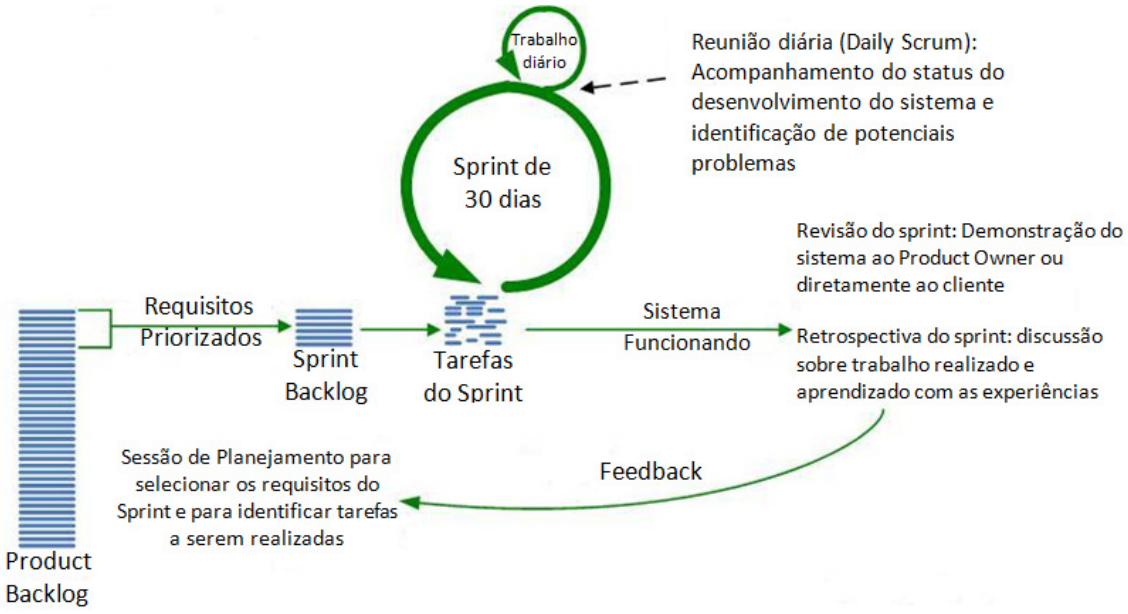


Figura 4.3 – Ciclo de vida de desenvolvimento do Scrum (adaptado de (AMBLER, 2010b))

O Scrum define três papéis principais: *Product Owner*, responsável pelos requisitos; Team, representado pelos desenvolvedores; e *Scrum Master*, representado pelo gerente. Os dois artefatos principais do Scrum são: *Product Backlog* – lista de requisitos que devem ser implementados no sistema e o *Sprint Backlog* – lista de tarefas que devem ser realizadas no *Sprint* e que podem ser visualizados na Figura 4.3.

O processo tem início quando o *Product Owner* tem a descrição geral do produto que será desenvolvido. A partir dessa descrição, denominada Visão, o *Product Backlog* é criado. No início de cada *Sprint* é realizado o *Sprint Planning Meeting* no qual o *Product Owner* prioriza o *Product Backlog*. Com base nessa priorização o *Team* seleciona as tarefas que irão compor o *Sprint Backlog*.

Durante o *Sprint*, a equipe de desenvolvimento realiza o *Daily Scrum* ou Reunião Diária, com duração de 15 minutos, na qual o trabalho realizado é sincronizado e os possíveis impedimentos são discutidos. Ao final de cada *Sprint* o time apresenta no *Sprint Review Meeting* a funcionalidade concluída e o *Scrum Master* encoraja o time a revisar o processo de desenvolvimento para torná-lo mais eficiente no próximo *Sprint*.

4.3 Planejamento de software no contexto ágil

Dentre os métodos apresentados na seção anterior, o Scrum destaca-se dos demais métodos pela ênfase dada ao gerenciamento de projeto. Na realidade, ele é um framework ágil que fornece um conjunto de boas práticas para atingir o sucesso de um projeto, auxiliando na construção iterativa de incrementos de um produto de software. O Scrum não define o que deve ser feito em todas as circunstâncias, podendo ser utilizado em projetos complexos nos quais não é possível prever tudo o que irá ocorrer (SCHWABER, 2004).

A abordagem ágil aplicada à gerência de projetos vem sendo discutida bem antes do surgimento do Scrum e vem ganhando força desde 2001, data da publicação do Manifesto do Desenvolvimento Ágil de Software (MANIFESTO, 2001).

Segundo Coram e Bohner (2005) os métodos ágeis são caracterizados por dar pouca ênfase ao planejamento formal, o que não quer dizer que o planejamento não seja realizado. Seja por meio de reuniões rápidas ou por meio de sessões mais longas de discussão, o trabalho a ser feito é planejado.

No estudo realizado por Abrahamsson *et al.* (2002) e apresentado na seção anterior observou-se que quase todos os métodos mencionados se preocupam com o planejamento e gerenciamento de projeto. Segundo Boehm (2002), comparando-se com o desenvolvimento ad-hoc, os métodos ágeis enfatizam o planejamento. Entretanto, por possuírem em seus princípios uma menor preocupação com a documentação, o produto do planejamento realizado nos métodos ágeis pode não ser visível facilmente.

O planejamento nos métodos ágeis é, em geral, menos extenso e realizado de forma mais rápida. Além disso, enquanto nos métodos tradicionais o controle é bastante formalizado, resultando em uma grande documentação, nos métodos ágeis o contato constante entre cliente e equipe torna o processo menos formalizado e, portanto, mais flexível. Como possui entregas realizadas em um tempo menor, pode parecer que o planejamento realizado no contexto ágil não atende aos prazos estabelecidos pelo cliente. No entanto, como o cliente participa da priorização dos requisitos e validação do trabalho, os riscos e possíveis problemas associados ao desenvolvimento são mais facilmente resolvidos, permitindo que os prazos estabelecidos possam, de maneira geral, ser cumpridos.

Um ponto chave para que os métodos ágeis funcionem na prática é o planejamento adequado das iterações. Esse planejamento deve ser baseado em estimativas do tamanho dos itens que serão desenvolvidos e do nível de produtividade dos membros da equipe de desenvolvimento.

Apesar dos itens de desenvolvimento possuírem diferentes formas de representação, as mais conhecidas são as Histórias do Usuário, que são uma breve descrição da funcionalidade a ser desenvolvida, de acordo com a visão do cliente do projeto (COHN, 2005), e o *Backlog Item* utilizado no Scrum (SCHWABER, 2004).

No capítulo 2 foram apresentadas algumas métricas para realização das estimativas, como a APF e o PCU. Dentre as propostas existentes na literatura para estimar o tamanho do trabalho a ser realizado nos métodos ágeis Cohn (2005) cita as seguintes:

- Pontos por História: unidade de medida que expressa o tamanho de uma história de usuário, ou de uma funcionalidade do sistema, ou mesmo de qualquer pedaço de trabalho que será implementado.
- Dias ideais: unidade de medida que corresponde a um dia ideal de trabalho no qual toda a estrutura disponível para o desenvolvimento estaria à mão e não ocorreria nenhuma interrupção.

Duas escalas de magnitude são sugeridas por Cohn (2005) para caracterizar a complexidade (ou o tamanho) do trabalho a ser realizado: a escala de Fibonacci, em que o próximo número da sequência corresponde à soma dos números anteriores (1, 2, 3, 5, 8,...), e uma variação dela, na qual os números da sequência correspondem à multiplicação do anterior por 2 (1, 2, 4, 8, 16,...). Essas escalas podem ser utilizadas em conjunto com o que Cohn denomina Planning *Poker*. Nessa forma de estimar a complexidade do trabalho, os membros da equipe recebem cartões com esses números e, para cada pedaço de trabalho, esses valores são lançados até que se chegue num consenso.

A partir da complexidade do trabalho estabelecida, é possível determinar a medida de velocidade (*velocity*) para estimar o trabalho a ser realizado com base nos esforços empregados anteriormente. A velocidade é uma medida empregada nos métodos ágeis que determina a quantidade de software que a equipe consegue entregar por iteração. Por meio dessa medida é possível planejar as próximas iterações e também acompanhar o desenvolvimento do projeto (KNIBERG, 2010).

4.4 VV&T no contexto ágil

Uma das principais características das metodologias ágeis é o contato direto e constante entre os membros da equipe de desenvolvimento e destes com o cliente. Dessa forma, o risco de construir um produto que não corresponda às necessidades do cliente é reduzido e os defeitos são detectados mais facilmente.

Ambler (2010a) ressalta duas práticas importantes para obter sucesso com a atividade de testes: considerar a testabilidade do software, ou seja, prever como o software deverá ser testado e comprovar o software desenvolvido constantemente de forma a obter a opinião do cliente, ou de seu representante, verificando se as funcionalidades entregues fazem o que deveriam fazer.

Segundo Opelt e Beeson (2008), um importante passo para conseguir aplicar as atividades de garantia da qualidade é conseguir que tanto os integrantes da equipe de desenvolvimento como os responsáveis pelo negócio reconheçam que existirão benefícios para o projeto. Dentre os benefícios estão: a identificação de defeitos antes que o sistema entre em produção, a adição de uma perspectiva diferente durante o desenvolvimento mais focada na qualidade do produto; e a percepção de quais são os defeitos mais críticos e que devem ser corrigidos.

Nos métodos tradicionais de desenvolvimento, como discutido no capítulo 3, a qualidade tem sido conseguida pelo uso de vários métodos de verificação e validação de software. Na prática, o ciclo de desenvolvimento dos métodos tradicionais prevê a realização de testes, entretanto, mesmo que especificados juntamente com os requisitos, geralmente são executados nas fases finais do ciclo de desenvolvimento. Nos métodos ágeis, de maneira geral, a garantia da qualidade do software recai no teste do código construído e na interação freqüente entre cliente e equipe de desenvolvimento. O desenvolvimento é realizado e versões são liberadas para serem validadas pelo cliente (HEDBERG; IISAKKA, 2006).

Os responsáveis pela realização dos testes nos métodos ágeis geralmente fazem parte da equipe de desenvolvimento e, dessa forma, o time como um todo é responsável pela qualidade do produto desenvolvido. Esta situação pode não se aplicar em ocasiões onde o ambiente de desenvolvimento seja mais complexo e surja a necessidade de existir uma equipe independente de testes que irá trabalhar em paralelo à equipe de desenvolvimento (AMBLER, 2010a).

As principais atividades de teste no contexto ágil estão relacionadas ao teste de unidade, difundidos pela utilização do TDD (*Test-driven development* ou *Test-driven design*). O TDD é uma prática adotada no desenvolvimento ágil que combina refatoração e o TFD (*Test-first development*). A refatoração implica em fazer pequenas modificações no código fonte existente de forma a melhorar o código sem necessariamente modificar a sua semântica. O TFD consiste em escrever um teste que determine o resultado de uma determinada funcionalidade e escrever código suficiente para este teste passar (BECK, 2004).

Podem existir dois níveis do TDD, o *Acceptance TDD*, onde os princípios do TDD são aplicados tendo como entrada os requisitos do sistema, e o Developer TDD, onde os testes são criados baseados na implementação que deverá ser realizada.

O *Acceptance TDD* é equivalente ao teste funcional e de aceitação realizado nos métodos tradicionais. Os testes de aceitação são realizados com base nas regras de negócio do sistema, ou seja, nos requisitos funcionais. Eles procuram verificar se o software está funcionando corretamente, se seu comportamento está conforme foi definido nas histórias do usuário ou requisitos. Estes testes devem ser realizados a cada entrega de um incremento de software, ou mesmo durante a iteração caso haja alguma incremente de software funcionando.

Segundo Ambler (2010a), a maior quantidade de testes deve ocorrer durante as iterações de desenvolvimento ou construção do software. Testes confirmatórios devem ser realizados pelo time de desenvolvimento e testes investigativos devem ser realizados de preferência, de forma paralela, por um time independente de testes. Os testes confirmatórios possuem o objetivo de verificar se o sistema cumpre o que os stakeholders definiram e o teste investigativo procura encontrar defeitos e problemas que o time de desenvolvimento pode não ter considerado.

O teste confirmatório é composto pelos testes de aceitação e pelos testes do “desenvolvedor”, que representam o teste efetuado sobre o código desenvolvido. Os testes de aceitação combinam os testes funcionais e de aceitação tradicionais. O teste de desenvolvedor une os testes unitários e de integração tradicionais. Unindo essas abordagens, o objetivo dos testes confirmatórios é de procurar defeitos no sistema, alcançando uma cobertura de código satisfatória e garantindo que o sistema está de acordo com os interesses dos *stakeholders*.

O teste investigativo procura explorar o que pode dar errado no sistema explorando cenários de teste que podem não ter sido explorados pela equipe de desenvolvimento. Os membros da equipe individual de testes, denominados testadores, descrevem os problemas encontrados sob a forma de defeitos, denominados "histórias defeito". Para corrigir um defeito a equipe de desenvolvimento deve tratá-lo como um requisito do sistema que será inserido no ciclo de desenvolvimento para correção. Os testes investigativos verificarão problemas de performance, integração, segurança, usabilidade, etc.

Uma pesquisa realizada pela empresa Ambysoft no ano de 2008 (AMBYSOFT, 2010) procurou sumarizar entre a comunidade que utiliza TDD quais são as técnicas de teste que eles utilizam na prática. Os resultados podem ser visualizados na Figura 4.4, onde pode-se notar que além de aplicar o TDD existe também a aplicação de atividades de revisão e inspeção, testes realizados no fim do ciclo de desenvolvimento e atividades paralelas de testes.

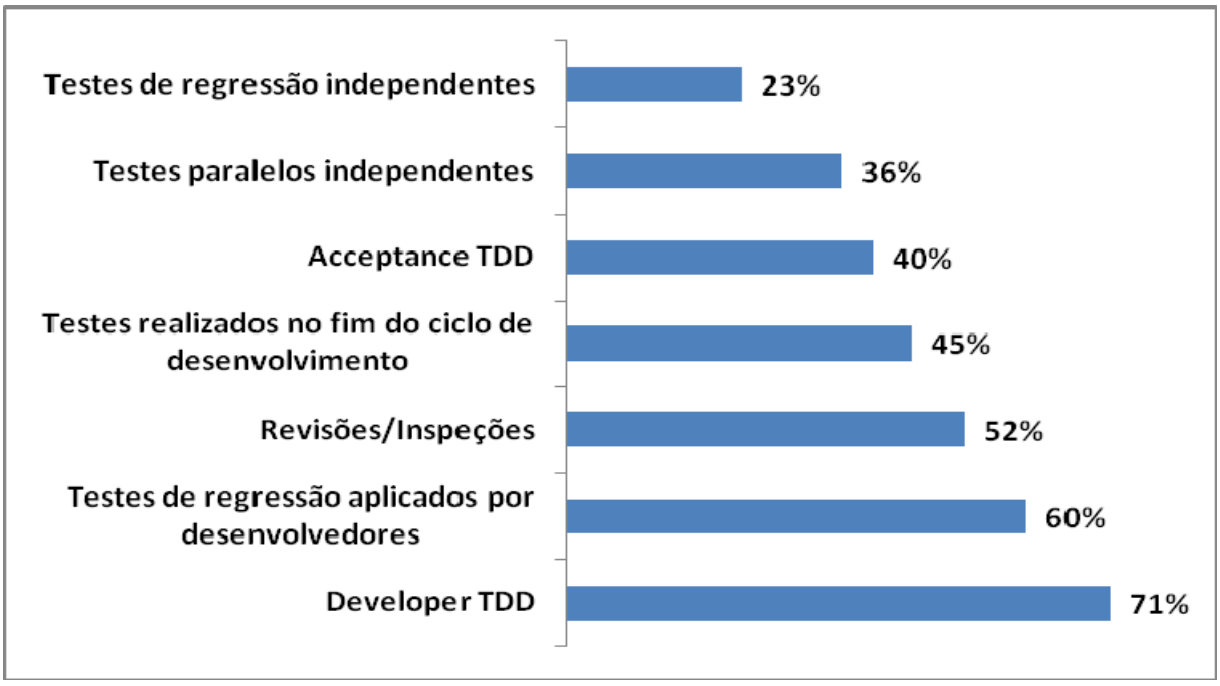


Figura 4.4 – Práticas de testes e validação de software executadas por desenvolvedores ágeis (adaptado de (AMBYSOFT, 2010))

4.5 Ferramentas de apoio à gerência e planejamento de projetos

Como observado na seção 4.3 os métodos ágeis podem divergir em algumas características o que permite determinar o perfil de cada método e as ocasiões nas quais ele pode ser aplicado. Essa diferenciação ocorre principalmente pela importância que um determinado método dá às fases do desenvolvimento de software. A maior importância ao processo prático de desenvolvimento faz com o que o método possua mais recomendações nessa área como é o caso do XP. Já no caso do Scrum, a maior importância é dada ao processo de gerenciamento do software.

Dessa forma, escolher uma ferramenta que auxilie a aplicação do método empregado pode facilitar bastante a aplicação das suas recomendações. Cabe a equipe de desenvolvimento empregar corretamente os recursos disponíveis na ferramenta de forma a tornar visíveis as atividades desenvolvidas pela equipe.

Segundo Pressman (2006), as ferramentas de gerência e planejamento de projetos devem apoiar o planejamento, acompanhamento e controle de um projeto. As informações do projeto devem, dessa forma, estar registradas em uma base comum onde tanto o gerente como a equipe de desenvolvimento possam acompanhar o progresso do projeto por meio de gráficos e medidas coletadas.

No contexto ágil, é importante que a ferramenta empregada possa facilitar o uso do método ágil escolhido e tornar o processo menos burocrático quanto possível. Na área de planejamento e acompanhamento do desenvolvimento do software, tais ferramentas devem prover a coleta e manutenção de dados que auxiliem a tomar decisões sobre o desenvolvimento do projeto.

Com características diferentes dos modelos de processo clássicos, como cascata e espiral, os métodos ágeis tem fomentado o desenvolvimento de soluções que se ajustem às suas necessidades especificas. A corrida comercial cresce nesse sentido, visando integrar ferramentas que suportem a aplicação dos métodos ágeis, e que possam se tornar acessíveis a um baixo custo a qualquer empresa de desenvolvimento de software.

Algumas ferramentas livres têm sido desenvolvidas de modo a auxiliar o acompanhamento de projetos ágeis. Dentre as principais iniciativas pode-se citar

Agilofant[3], XPlanner[4], TargetProcess[5], Agilo for Scrum[6], IceScrum[7], FireScrum[8], etc.. Além destas, existem ferramentas que mesmo sendo pagas são disponibilizadas para uso com menos funcionalidades como, por exemplo, a ScrumWorks (DANUBE, 2010). Tais ferramentas buscam fornecer a possibilidade de gerenciar um projeto seguindo o modelo ágil onde a documentação tende a ser reduzida e as entregas são feitas em períodos mais curtos e de forma incremental.

Apesar de serem criadas por grupos diferentes e desenvolvidas para suprir determinada demanda de mercado, se corretamente agrupadas, as ferramentas open source podem suportar o ciclo de desenvolvimento de uma empresa sem que ela tenha grandes custos. É importante observar o processo de desenvolvimento de cada empresa para perceber onde e quando as ferramentas existentes podem auxiliar na melhoria do processo.

Dentre as ferramentas citadas, duas ferramentas foram escolhidas para exemplificar as características de ferramentas de planejamento e gerenciamento de projetos, a ScrumWorks, desenvolvida pela Danube Tecnologies e já bastante consolidada na área, e a FireScrum, ferramenta freeware e *open source*, desenvolvida por um grupo de estudantes voluntários de Engenharia de Software da Universidade Federal de Pernambuco (UFPE). Tais ferramentas serão apresentadas nas seções seguintes.

4.5.1 ScrumWorks

A ScrumWorks é uma ferramenta desenvolvida pela *Danube Technologies* que possui uma versão comercial (*Pro Edition*) e uma versão gratuita (*Basic Edition*) com recursos mais básicos (DANUBE, 2010).

A *ScrumWorks Basic Edition* auxilia o uso do framework Scrum durante o ciclo de desenvolvimento de um produto permitindo aos usuários gerenciar o ciclo de entregas e monitorar o progresso do trabalho realizado durante um Sprint. Dentre as suas principais características estão a interface *drag-and-drop* e intuitiva que permite arrastar as informações exibidas na tela e organizadas em duas colunas. Nas duas

[3] http://www.agilefant.org/
[4] http://www.xplanner.org/
[5] http://www.targetprocess.com/
[6] http://www.agile42.com/cms/pages/agilo/
[7] http://www.icescrum.org/
[8] http://www.firescrum.com/

colunas exibidas é possível acompanhar o trabalho que já foi realizado (*Commited Backlog*) e o trabalho que ainda será realizado (*Uncommited Backlog*) como pode ser observado na Figura 4.5 – Tela principal da ferramenta ScrumWorks (DANUBE, 2010). Além disso, os usuários podem realizar o gerenciamento de entregas, com a definição de cronogramas para entrega do produto de trabalho desenvolvido.

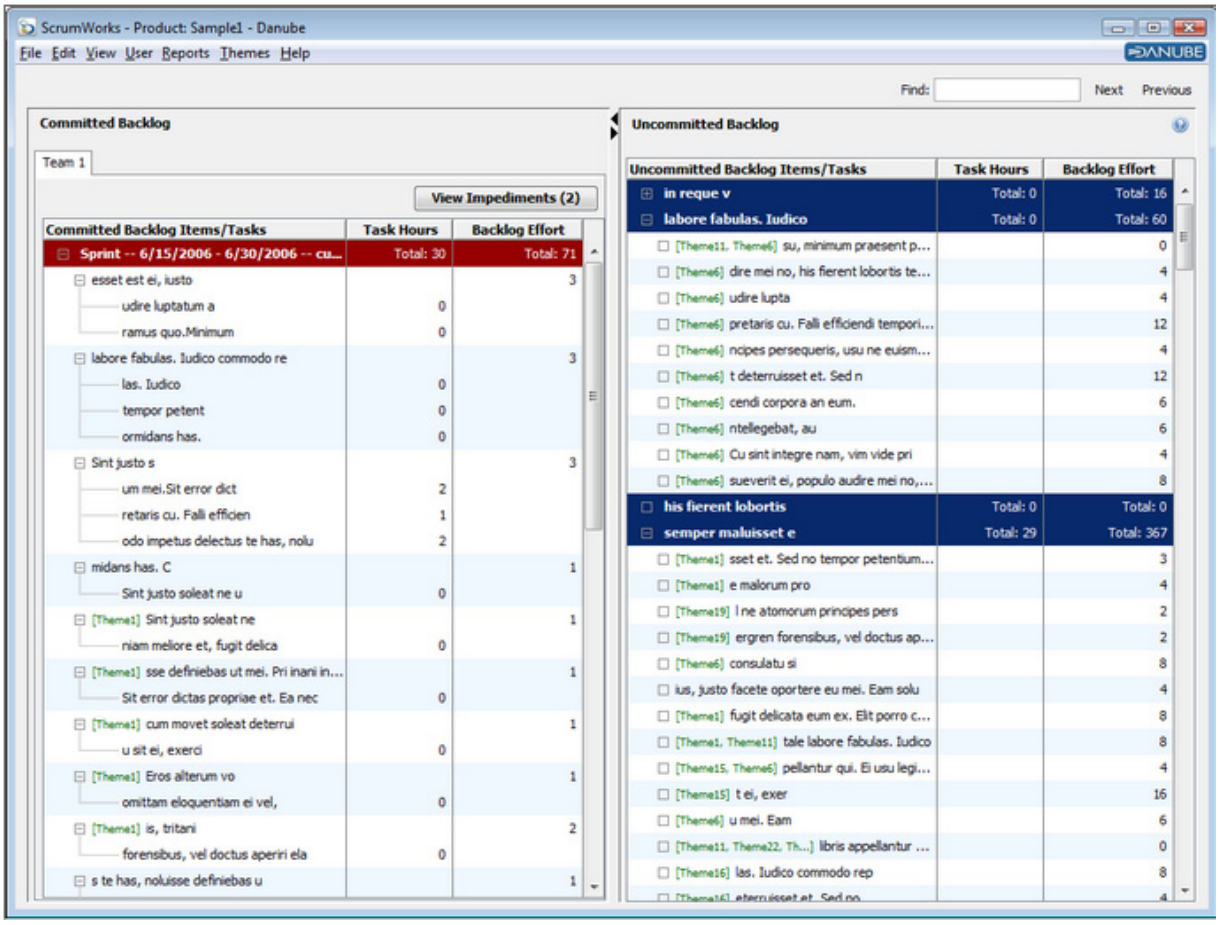


Figura 4.5 – Tela principal da ferramenta ScrumWorks (DANUBE, 2010)

A ScrumWorks permite também a geração de relatórios que se baseiam nas necessidades do Scrum (Figura 4.6). A ferramenta permite saber algumas métricas como velocidade da equipe, a taxa de entregas do software e a taxa de mudanças nos requisitos. Tais informações podem ser utilizadas para gerar relatórios customizados e estimativas mais próximas da realidade da equipe.

A ScrumWorks prove também uma API que pode ser utilizada para construir relatórios e integrar a ferramenta com outras aplicações.

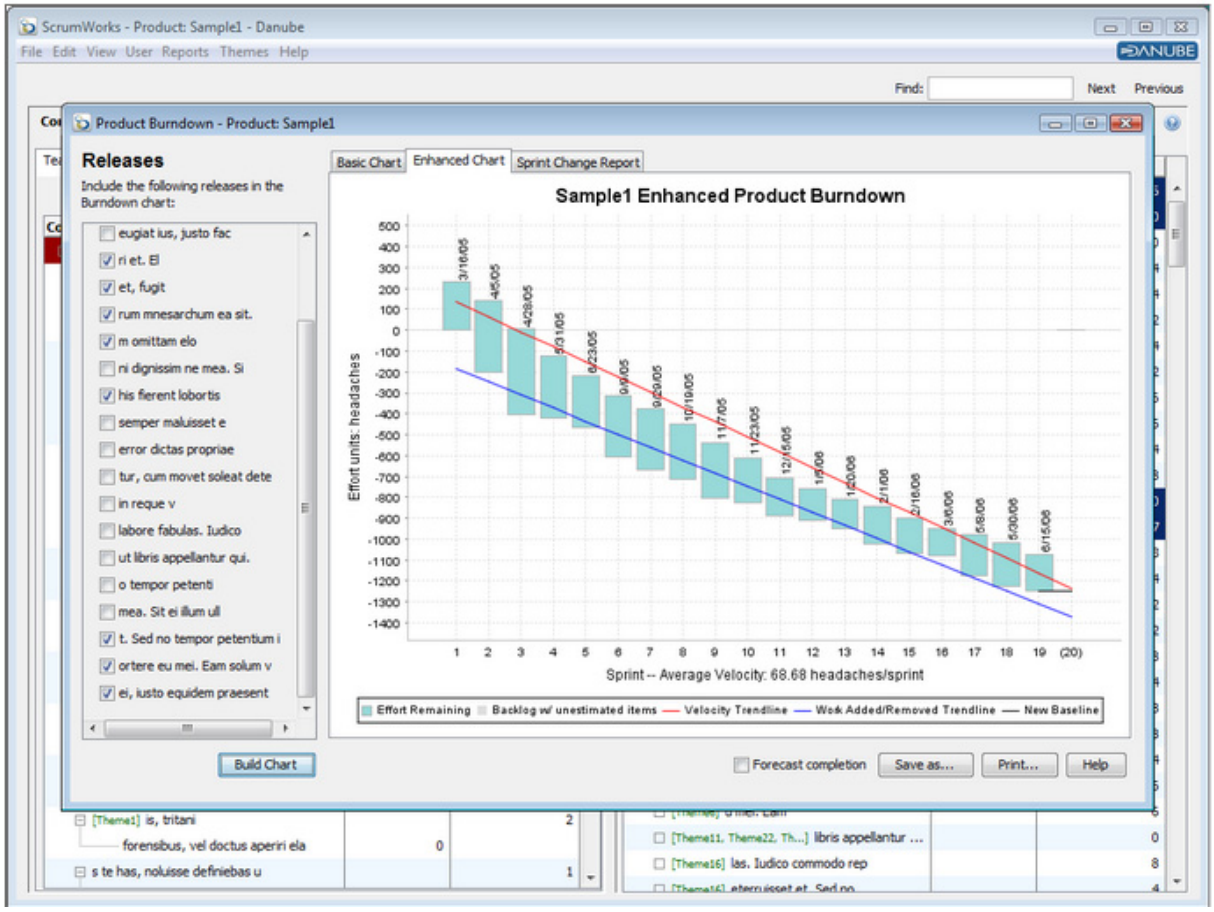


Figura 4.6 – Geração de relatórios na ferramenta ScrumWorks (DANUBE, 2010)

4.5.2 FireScrum

O FireScrum é uma ferramenta desenvolvida utilizando conceitos da Web 2.0 e de *Rich Internet Application* que reúne um conjunto de aplicações integradas para suportar equipes que utilizam o Scrum como base para o desenvolvimento de seus projetos, sendo especialmente útil para equipes que trabalham remotamente (CAVALCANTI, MACIEL, ALBUQUERQUE, 2009).

O objetivo principal da ferramenta FireScrum é de apoiar equipes que utilizam o Scrum contudo, possui alguns módulos de apoio que foram criados para concentrar outras necessidades existentes no gerenciamento de um projeto, excluindo a necessidade de ferramentas terceirizadas, como o gerenciamento de testes e defeitos. Os módulos do FireScrum são organizados em Core, Taskboard, Planning Poker, Bug Tracking e Desktop Agent.

O módulo Core é responsável pela parte operacional básica do Scrum por meio do qual é possível realizar as seguintes operações: controle de acesso a ferramenta, cadastramento de usuários, cadastramento de projetos, criação de itens de backlog, priorização de itens de backlog, criação de sprints, associação de itens de backlog a sprints, criação de tarefas para um dado item de backlog, alocação do membro do time a determinada tarefa, geração do gráfico burndown da sprint e gráfico de burndown do produto. Na Figura 4.7 – Módulo Core da ferramenta FireScrum é possível visualizar a tela do módulo Core onde é possível criar os itens e tarefas a serem desenvolvidos e os sprints necessários no desenvolvimento do projeto. Assim como a ferramenta ScrumWorks apresentada na seção anterior, a interface é *drag-and-drop* e para mover um item de uma coluna a outra é necessário arrastá-lo.

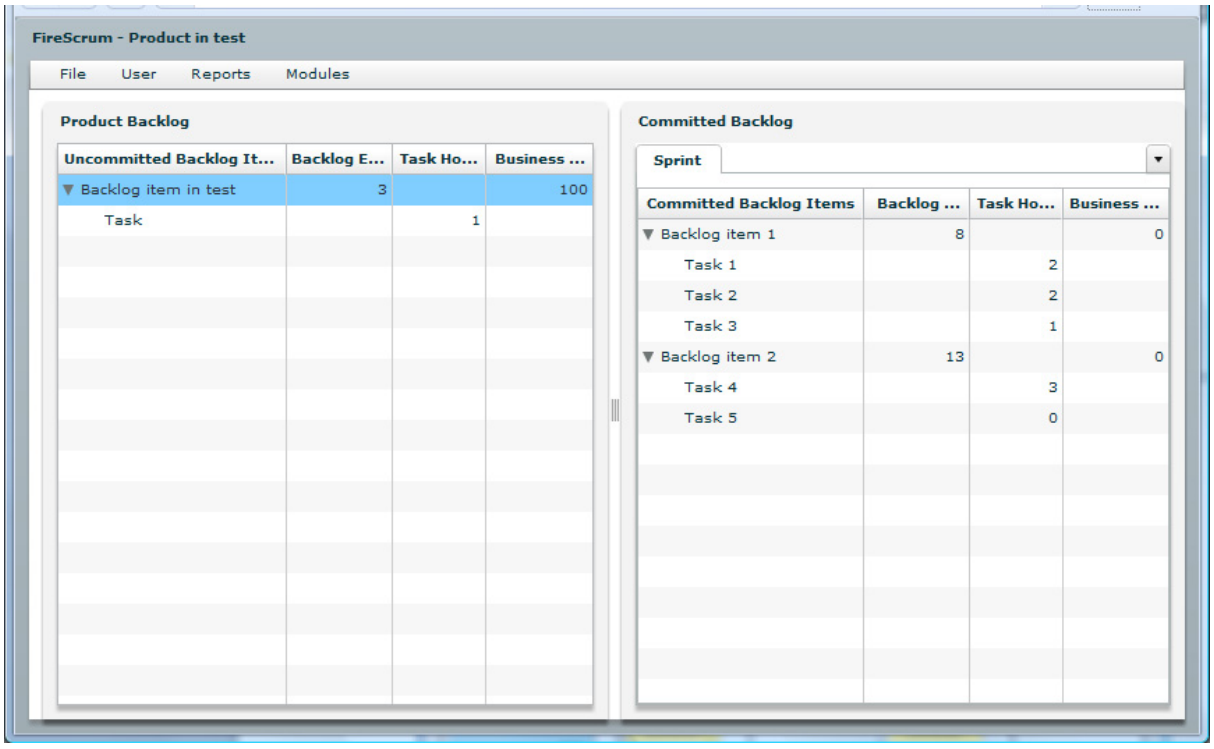


Figura 4.7 – Módulo Core da ferramenta FireScrum

O módulo *Taskboard* reproduz o quadro físico ou mural utilizado em projetos reais baseados em *Scrum*, onde os usuários podem manipular *post-its* virtuais que contém as tarefas a ser realizadas. Cada item inserido no *sprint* é mostrado na tela no campo *Backlog Item*, e as tarefas são representadas por "*post-its*" que podem ser movidos entre os seguintes campos: *To Do* que representa as tarefas a fazer; *Impeded* que representa as tarefas que estão impedidas de serem realizadas por

algum motivo; *In Progress* que representa as tarefas em progesso, em andamento; e o *Done* que representa as tarefas que já foram concluídas. Como pode ser visualizado na Figura 4.8, para cada tarefa é exibido também o nome atribuído a tarefa e o responsável pela sua realização.

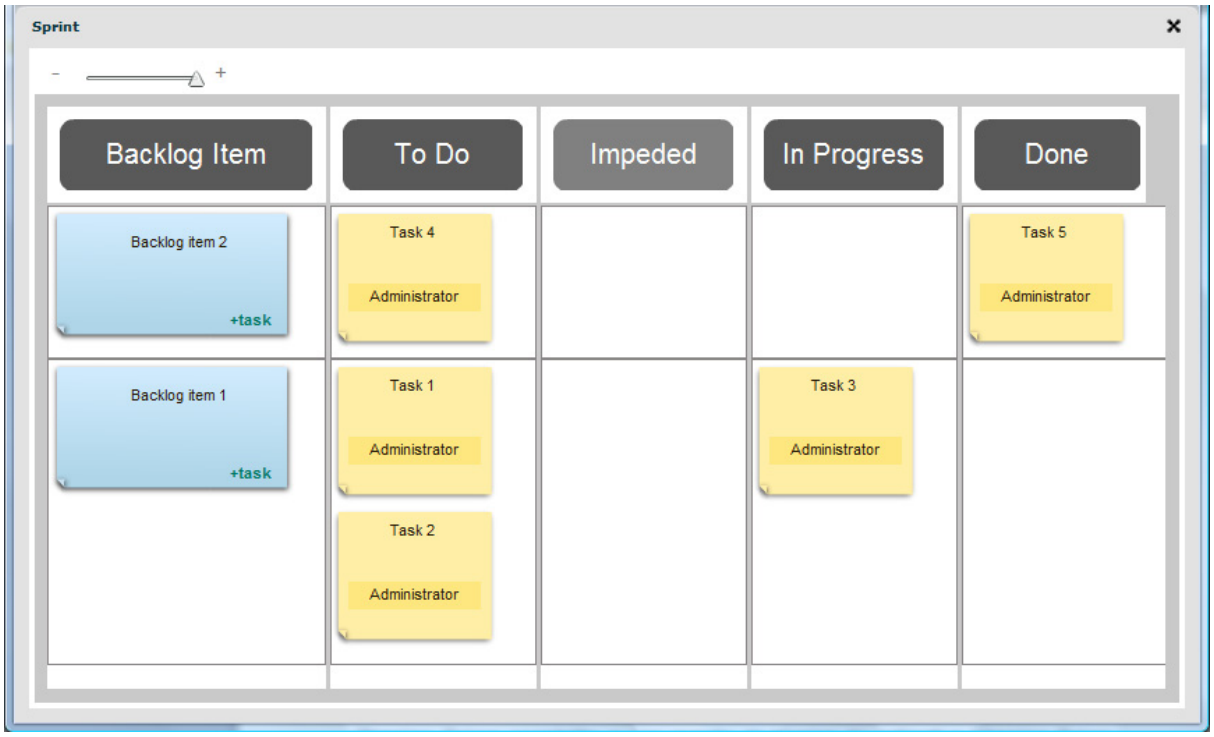


Figura 4.8 – Módulo Taskboard da ferramenta FireScrum

O módulo *Planning Poker* possibilita o uso remoto da técnica de estimativa *Planning Poker* por meio do uso de *chat*, vídeo ou texto. Assim, mesmo a equipe estando separada geograficamente é possível realizar as sessões de planejamento com todos os membros da equipe como pode ser visualizado na Figura 4.9.

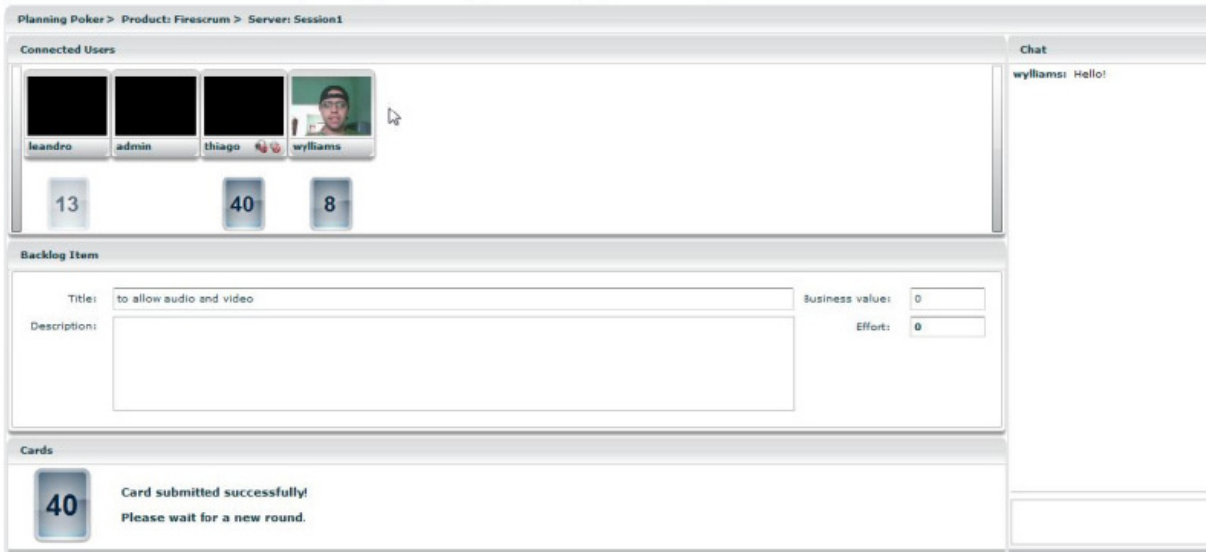


Figura 4.9 – Módulo *Planning Poker* da ferramenta FireScrum (CAVALCANTI;MACIEL;ALBUQUERQUE, 2009)

O módulo *Test Management* permite a criação e gerenciamento de casos de testes e dentre as funcionalidades disponíveis neste módulo destacam-se: criação de plano de teste, criação de casos de teste, associação de casos de teste a itens do *backlog*, registro dos resultados dos testes. Na Figura 4.10, pode-se visualizar a tela do módulo *Test Management*. Para criar um caso de teste é necessário criar uma suíte de testes (*Test Suite)* e para executar os testes criados é necessário criar um plano de teste (*Test Plan*).

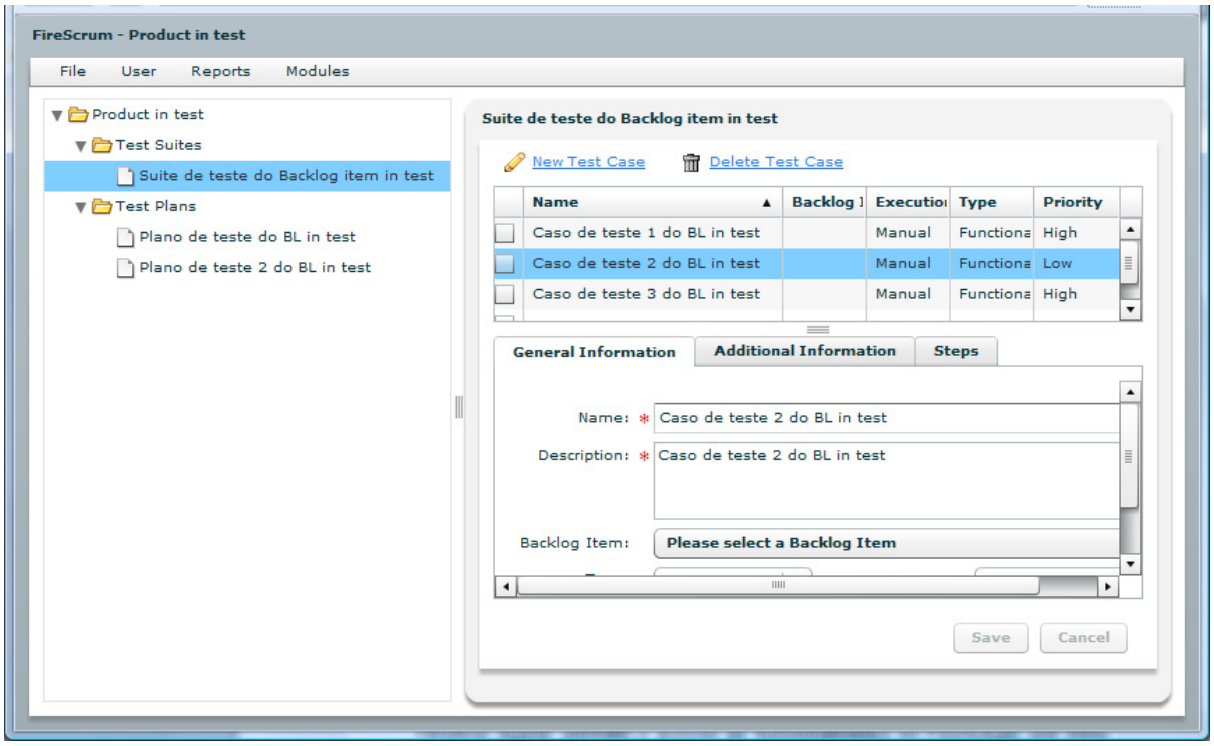


Figura 4.10 – Módulo Test Management da ferramenta FireScrum

O módulo de *Bug Tracking* (Figura 4.11) fornece recursos para o registro de defeitos e todo o ciclo de vida necessário para a sua resolução. As funcionalidades disponíveis neste módulo são o registro de defeito, a associação de um item de *backlog* a um defeito, a definição de um responsável pelo defeito; além disso, permite adicionar notas ao defeito, anexar arquivos com evidências do problema, manter histórico e gerar relatórios com filtros sobre defeitos.

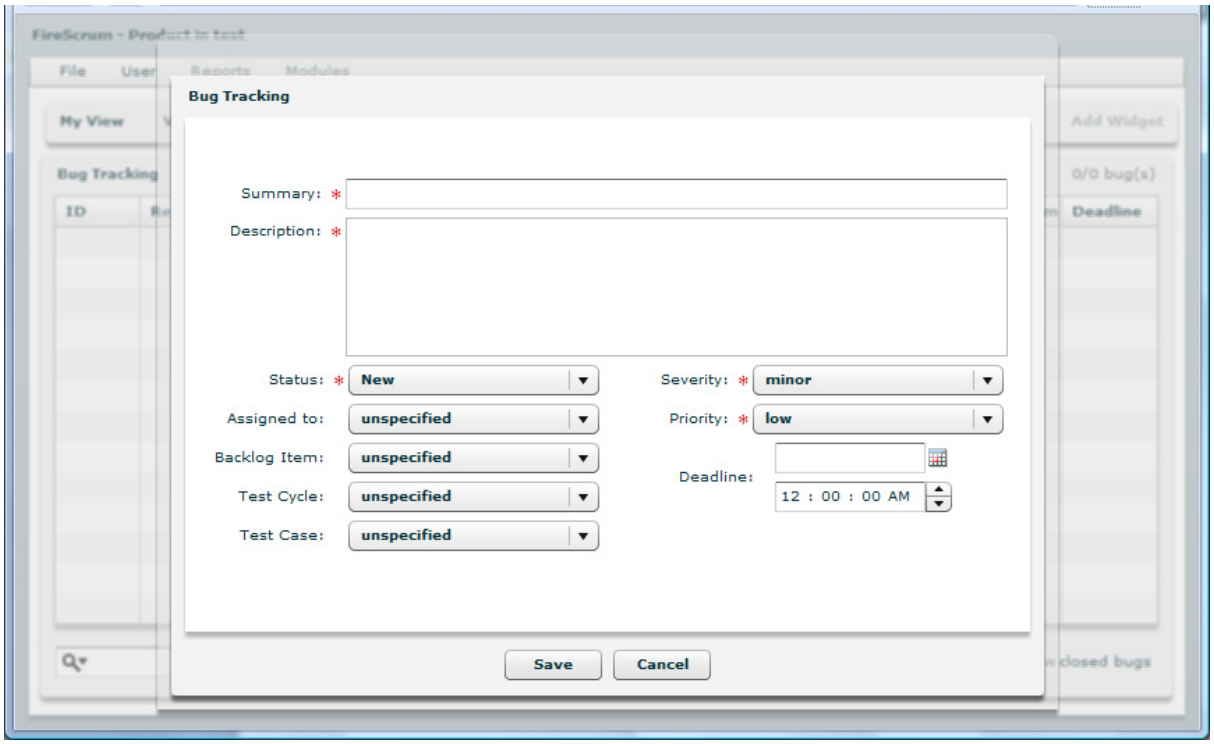


Figura 4.11 – Módulo *Bug Tracking* da ferramenta FireScrum

O *Desktop Agent* permite o acesso as funcionalidades do FireScrum, por meio do acesso a menu disponível no *system tray* do Sistema Operacional. Por meio do *Desktop Agent* é possível visualizar e editar tarefas, receber alertas do sistema, reportar defeitos e comunicar com outros usuários do FireScrum por meio de *chat*.

A arquitetura funcional do FireScrum foi obtida por meio de uma análise de algumas das ferramentas mais conhecidas na comunidade para identificar lacunas e potenciais diferenciais de mercado. As ferramentas analisadas foram VersionOne[9], TargetProcess, ScrumWorks e Agilo for Scrum. Na Tabela 4.2 é possível visualizar a comparação realizada por Cavalcanti, Maciel e Albuquerque (2009) das

[9] http://www.versionone.com/

funcionalidades existentes nas ferramentas analisadas em comparação ao FireScrum.

Tabela 4.2 – Analise comparativa de funcionalidades (adaptado de (CAVALCANTI, MACIEL, ALBUQUERQUE, 2009))

Funcionalidades	Version One	Target Process	ScrumWorks	FireScrum	Agilo for Scrum
Product Management	X	X	X	X	X
Sprint Management	X	X	X	X	X
Report and Analytics	X	X	X	X	X
User Management	X	X	X	X	X
Web Services API	X	X	X	X	--
Bug Tracking	X	X	--	X	X
Taskboard	X	X	X	X	--
Test Management	--	X	--	X	--
Sprint Review	X	--	--	--	--
Sprint Retrospective	X	--	--	--	--
Release Planning	X	X	X	X	X
Impediments Management	--	X	X	X	X
Desktop Agent	--	--	--	X	--
Planning Poker	--	--	--	X	--

É possível perceber por meio da comparação realizada que excetuando as atividades *Sprint Review* e *Sprint Retrospective*, a FireScrum provê suporte a todas as outras atividades com o acréscimo de permitir a realização do *Planning Poker* utilizando recursos multimídia e também o *Desktop Agent* que permite acessar as funcionalidades da ferramenta por meio do *system tray*. Dessa forma, por ser uma ferramenta *opensource* se torna acessível a empresas de desenvolvimento de software, em especial as pequenas e médias empresas, que não possuem recursos financeiros suficientes para investir em ferramentas que auxiliem no processo de desenvolvimento de software.

4.6 Considerações finais

Neste capítulo foram apresentados as características das metodologias ágeis, seu ciclo de vida e os métodos ágeis mais utilizados atualmente. Foram discutidas

também as abordagens ágeis de gerência e planejamento de projetos bem como de VV&T que podem ser aplicados por empresas que tenham escolhido seguir o caminho do desenvolvimento ágil. Foram também apresentadas algumas ferramentas existentes que podem auxiliar na realização do planejamento e gerenciamento de projetos. Dentre as ferramentas levantadas foram mostradas as ferramentas ScrumWorks e FireScrum. Pela análise realizada foi possível perceber que a ferramenta FireScrum desponta como uma boa opção para empresas de pequeno e médio porte que queiram aplicar os conceitos ágeis sem implicar em grandes gastos.

Com relação ao planejamento de software, independentemente da técnica utilizada para realizar a estimativa é importante saber como realizar o controle dessa estimativa e planejamento. Além disso, para que a qualidade do produto de software seja alcançada é necessário estabelecer um conjunto de atividades de teste que devem ser realizadas pela equipe de desenvolvimento, ou mesmo por uma equipe independente de testes, para validar continuamente o produto desenvolvido.

De acordo com essas observações, foi desenvolvida a estratégia PW-PlanTe apresentada no próximo capítulo. Essa estratégia tem por objetivo auxiliar na realização do planejamento e controle do planejamento de software, bem como estabelecer atividades de teste de software que permitam garantir minimamente a qualidade do produto de software desenvolvido. Por ser baseada em iterações ela pode ser adotada juntamente com o uso de métodos ágeis e por trabalhar com *Pieces of Work* (Pedaços de Trabalho) em cada iteração, permite que várias técnicas de estimativas possam ser utilizadas.

Capítulo 5

ESTRATÉGIA PW-PLANTE

Neste capítulo é apresentada a estratégia PW-PlanTe, uma evolução da estratégia PCU|$_{PSP}$, desenvolvida como forma de auxiliar a elaboração e acompanhamento das atividades de planejamento e a realização dos testes. Essa estratégia foi desenvolvida no contexto de uma empresa de pequeno porte, em que os prazos e orçamentos são reduzidos, e as entregas devem ser realizadas de forma rápida. Essa empresa adota os conceitos de métodos ágeis, pelo uso do framework Scrum e, sendo assim, o planejamento das iterações deve ser o mais preciso possível e a qualidade do produto desenvolvido deve ser garantida de forma a manter a empresa no mercado.

5.1 Considerações Iniciais

As atividades de planejamento e garantia da qualidade são partes importantíssimas do processo de desenvolvimento de software. Sua importância é ainda maior no contexto de desenvolvimento ágil de software, em que o produto é construído de forma iterativa e incremental. Nesse contexto o planejamento do trabalho a ser feito durante as iterações deve ser o mais preciso possível, considerando atividades de desenvolvimento e garantia da qualidade que, por sua vez, devem ser menos onerosas tanto com relação ao tempo quanto ao custo.

Como pode ser observado nos capítulos anteriores, na literatura são citadas técnicas que auxiliam na previsão e estimativa do tamanho do software, como as tradicionais APF e PCU, e as técnicas mais recentes inseridas no contexto ágil, Pontos por História e Dias Ideais. Entretanto, na grande maioria das empresas o planejamento das iterações ainda permanece sendo realizado com base na experiência anterior de alguns membros da equipe.

As atividades de garantia da qualidade de software, em especial as atividades de VV&T, apesar de já bastante difundidas nos meios acadêmico e comercial, ainda são negligenciadas, especialmente no contexto de pequenas empresas de desenvolvimento de software. Isso ocorre principalmente pelo fato da quantidade de recursos humanos nesse tipo de empresa ser menor e os cronogramas extremamente reduzidos.

Para as empresas de pequeno porte que lidam diariamente com a pressão do mercado em entregar sistemas de qualidade e com prazos restritos, controlar prazos e custos é fundamental para conseguirem se manter no mercado. A receita financeira dessas empresas está diretamente ligada à entrega de cada sistema dentro dos acordos assumidos.

Considerando esse cenário, conforme apresentado no Capítulo 2, Sanchez (2008) propôs a estratégia PCU|$_{PSP}$com o objetivo de obter estimativas mais precisas combinando planejamento e controle. Após a definição da estratégia PCU|$_{PSP}$, pode-se notar a similaridade entre as suas etapas e as etapas do framework ágil Scrum, relacionando-se de forma bastante natural às práticas ágeis desse framework. O contexto de desenvolvimento da Linkway é, assim como no Scrum, composto por equipes pequenas de desenvolvimento que necessitam entregar e validar continuamente o software desenvolvido.

Dessa forma, com o objetivo de estabelecer uma maneira mais estruturada de realizar e acompanhar o planejamento de software, inserindo práticas ágeis e provendo recursos para que o software produzido tenha qualidade agregada, a estratégia PCU|$_{PSP}$ foi adequada, gerando a Estratégia PW-PlanTe. A PW-PlanTe considera técnicas alternativas de planejamento ágil, define um processo de teste e define também um conjunto de ferramentas livres para dar suporte às etapas que compõem a estratégia.

A estratégia PW-PlanTe auxilia o planejamento e acompanhamento de iterações com base nas estimativas calculadas utilizando a complexidade do trabalho a ser desenvolvido e o Nível de Esforço (NDE) empregado no desenvolvimento. Nesse aspecto ela é composta por atividades de planejamento e controle, que permitem verificar se o planejamento das atividades de desenvolvimento e garantia da qualidade estão sendo realizadas no período determinado para sua execução. Nela, as iterações podem ser estimadas não apenas por meio de casos de uso, como era feito na PCU|$_{PSP}$, mas por meio de

outras unidades de medida mais frequentemente adotadas no contexto ágil, como por exemplo, histórias do usuário, o que torna a estratégia mais genérica.

As atividades de garantia da qualidade de software, mais especificamente a atividade de teste, ausente na $PCU|_{PSP}$, foi acrescentada na PW-PlanTe por meio da inserção de um processo de teste que busca realizar, de forma prática e ágil, atividades de teste no ciclo de vida de desenvolvimento de software. Procurou-se estabelecer atividades de teste que não impactassem no tempo e custo dos projetos de uma forma que uma pequena empresa pudesse absorver, e que permitissem que essas empresas pudessem sair de um estado em que a garantia da qualidade é realizada de forma *ad-hoc*, sem nenhum planejamento, para um estado em que essas atividades possam ser realizadas mais formalmente.

Essa estratégia foi desenvolvida por meio de um processo constante de observação, criação e avaliação de soluções, denominado por Potts (1993) como Indústria como Laboratório (*Industry as Laboratory*). Esta abordagem de pesquisa procura descobrir o que realmente é possível realizar na prática. Assim, as idéias para a realização da pesquisa baseiam-se em problemas reais, que com o decorrer do tempo são refinados em estudos de caso executados de forma contínua e incremental. No caso deste trabalho, foi exatamente seguindo um processo como esse que a estratégia PW-PlanTe foi desenvolvida, estando constantemente em adaptação e evolução.

Nas próximas seções descreve-se o trabalho realizado, organizado da seguinte maneira: na Seção 5.2 apresenta-se a Estratégia PW-PlanTe, suas principais características e seu funcionamento; na Seção 5.3 é detalhado o processo de implantação da estratégia com o uso de softwares livres; na Seção 5.4 apresenta-se a relação entre a estratégia elaborada e o framework ágil Scrum; e por fim, na Seção 5.5 são apresentadas as considerações finais com as principais lições aprendidas no desenvolvimento dessa estratégia.

5.2 Definição da Estratégia PW-PlanTe

Sendo os métodos ágeis uma alternativa proposta para equipes pequenas e sistemas não muito complexos (BECK, 2004), eles se tornam apropriados no contexto de empresas de pequeno porte.

Dessa forma, considerando a constante iniciativa da empresa de pequeno porte participante deste trabalho em melhorar a qualidade de seu processo de desenvolvimento e a reestruturação do seu processo de forma a adotar o framework Scrum, o próximo passo de melhoria almejado era sistematizar a atividade de teste. Essa sistematização deveria considerar a aplicação das atividades de testes de forma combinada com a nova abordagem ágil de desenvolvimento proposta. Nesse contexto, deu-se início ao estudo e adaptação da $PCU|_{PSP}$ para contemplar esses novos objetivos. A estratégia $PCU|_{PSP}$, foi evoluída considerando os aspectos de planejamento e de garantia de qualidade de software, particularmente, a atividade de teste.

No que diz respeito ao planejamento, as técnicas de estimativa de tamanho mais comumente empregadas no contexto ágil foram estudadas e as iterações da estratégia foram adaptadas para a aplicação não somente dos pontos por casos de uso, como também para outras unidades de trabalho, como pontos por história. Essa adaptação gerou o nome da nova estratégia – PW-PlanTe (SANDE *et al.*, 2010), na qual o PW (*Piece of Work*) denomina a quantidade de trabalho que pode ser alocada em uma iteração.

As etapas da PW-PlanTe relacionam-se às práticas ágeis propostas no Scrum e, assim como na estratégia $PCU|_{PSP}$, a base para a coleta de medidas de tempo utilizadas para o estabelecimento de estimativas tem seu suporte na aplicação do PSP nível 1.1 e na ferramenta livre *Process Dashboard*. Entretanto, mesmo que uma empresa não utilize a *Dashboard* e não adote o PSP como modelo de qualidade individual de seus desenvolvedores, ainda assim a estratégia PW-PlanTe pode ser adotada, bastando para isso que o tempo gasto com as atividades desenvolvidas seja registrado.

Da mesma forma que na $PCU|_{PSP}$, a estratégia PW-PlanTe é composta por dois grandes blocos, o de planejamento e o de controle, que são constantemente executados, realimentando um ao outro por meio de atividades do PSP. As

atividades do PSP induzem a uma realimentação constante e ajuste de estimativas por meio do cálculo do Nível de Esforço (NDE) de cada desenvolvedor, que corresponde à relação entre tempo gasto e trabalho realizado. O tempo é controlado por meio da Dashboard e o trabalho pode ser caracterizado pela complexidade obtida por meio da técnica utilizada, que não está mais restrita à aplicação dos Pontos por Caso de Uso. No bloco de planejamento é possível determinar estimativas adequadas de tamanho, prazos e custos antes do desenvolvimento do sistema ou mesmo de uma determinada funcionalidade. No bloco de controle é avaliado se o planejamento determinado está sendo cumprido e, caso não esteja, o motivo desse fato. Por meio da realimentação entre esses dois blocos, o planejamento é constantemente ajustado.

Ao trabalho a ser realizado numa iteração é dado o nome de *Piece of Work* (PW) o qual irá variar de acordo com o método de estimativa utilizado para o sistema, que pode ser um conjunto de Casos de Uso, de Histórias do Usuário, etc. Assim, o “Pedaço de Trabalho” (PW) determinado para uma iteração é, em geral, composto de partes menores denominadas “Itens de Trabalho” (IW - *Item of Work*), que pode vir a ser um caso de uso ou uma história, e que, por sua vez, podem ser compostos por “Tarefas” (*Tasks*). A quebra de um “pedaço de trabalho” em “itens de trabalho” e “tarefas” permite uma aplicação sistemática do PSP e é justamente isso que possibilita com que as práticas de planejamento e controle desse modelo sejam efetivamente adotadas no dia-a-dia de uma empresa de pequeno porte. Assim, esse conjunto de atividades proposto na estratégia PW-PlanTe facilita a adoção de boas práticas de desenvolvimento e reduz um pouco a resistência à mudança cultural, que mesmo assim foi bastante presente durante o desenvolvimento deste trabalho.

Os papéis existentes na estratégia baseiam-se nos papéis existentes no Scrum e se relacionam da seguinte forma: o Scrum Master é representado pelo Gerente; a equipe de desenvolvimento, composta por desenvolvedores e testadores, corresponde ao Team; o Product Owner (PO) é o representante do cliente para a equipe de desenvolvimento, o qual é o responsável pelo retorno do investimento, pela priorização e pela aceitação dos IWs desenvolvidos.

As atividades de garantia da qualidade de software acrescentadas à estratégia PW-PlanTe foram selecionadas com o objetivo de que não causassem um grande impacto no esforço empregado pela equipe. As atividades selecionadas, que podem ser realizadas tanto de forma manual como automatizada, correspondem à

realização do planejamento e elaboração dos casos de teste, execução dos testes, registro e acompanhamento de defeitos, verificação da cobertura dos testes e geração de relatórios de evidência de cobertura. A inserção das atividades de teste fez com que fosse também introduzido na estratégia outro papel, o de testador. Esse papel representa a pessoa responsável pela realização das atividades de teste na equipe de desenvolvimento, garantindo assim que o produto desenvolvido tenha sido continuamente avaliado durante a iteração e que a possibilidade de existirem erros seja menor.

No contexto de empresas de pequeno porte, com número reduzido de recursos humanos disponíveis, caso não exista a possibilidade de ter uma pessoa na equipe dedicada à realização dos testes, as atividades de teste podem ser executadas pelos desenvolvedores.

Além das atividades de testes realizadas pelo testador sobre os incrementos de software produzidos durante a iteração, eles devem ser também continuamente validados com o representante do cliente na empresa, o PO, de forma que o produto desenvolvido seja realmente aquilo que o cliente solicitou. Essa atividade de aceitação do produto feita pelo PO deve ser realizada durante a iteração, pois caso sejam identificadas discrepâncias entre o que o cliente solicitou e o que foi realmente desenvolvido ainda há tempo de corrigir esses problemas sem despender esforço em retrabalho.

A Figura 5.1 apresenta a estratégia PW-PlanTe que em linhas gerais pode ser resumida da seguinte forma: as Etapas 1, 2 e 3 que compõem o bloco de planejamento têm por objetivo geral selecionar os IWs que compõem o sistema a ser desenvolvido, agregando a eles uma ordem de prioridade de desenvolvimento e uma medida de complexidade que caracterize quão difícil é a sua realização. Alguns desse IWs serão selecionados para compor a iteração e quebrados em Tarefas menores que serão estimadas com o tempo a ser gasto para o seu desenvolvimento. Para cada IW, durante a Etapa 4, deverão ser escritos testes de aceitação que após o desenvolvimento e conclusão de todas as Tarefas do IW (Etapa 5) serão executados (Etapa 7) permitindo avaliar se o funcionamento do IW está conforme o esperado e se as funcionalidades desenvolvidas foram satisfatoriamente exercitadas (Etapa 8).

No bloco de controle, durante a Etapa 6, ao finalizar cada Tarefa do IW desenvolvido de forma iterativa e incremental, o tempo gasto deve ser comparado ao

estimado de forma que seja possível identificar alguma discrepância e ajustar as estimativas das outras Tarefas deste IW. Quando um IW é concluído (Etapa 9), a produtividade da equipe deve ser analisada por meio do cálculo do NDE realizado na Etapa 11. Esse valor de NDE retrata a produtividade do time durante o desenvolvimento deste IW e deve ser utilizado para avaliar se o trabalho alocado para a iteração poderá ser realmente realizado ou serão necessários ajustes. Quando todos os IWs que compõem a iteração são concluídos (Etapa 10), o NDE deve ser mais uma vez calculado na Etapa 11 e avaliado durante a reunião de controle, realizada na Etapa 12. Nesse momento são discutidos os possíveis desvios na produtividade da equipe e após essa reunião, o fluxo de execução da PW-PlanTe retorna então à Etapa 2, na qual a capacidade de trabalho da próxima iteração será recalculada com base nesse novo NDE que representa a produtividade real da equipe.

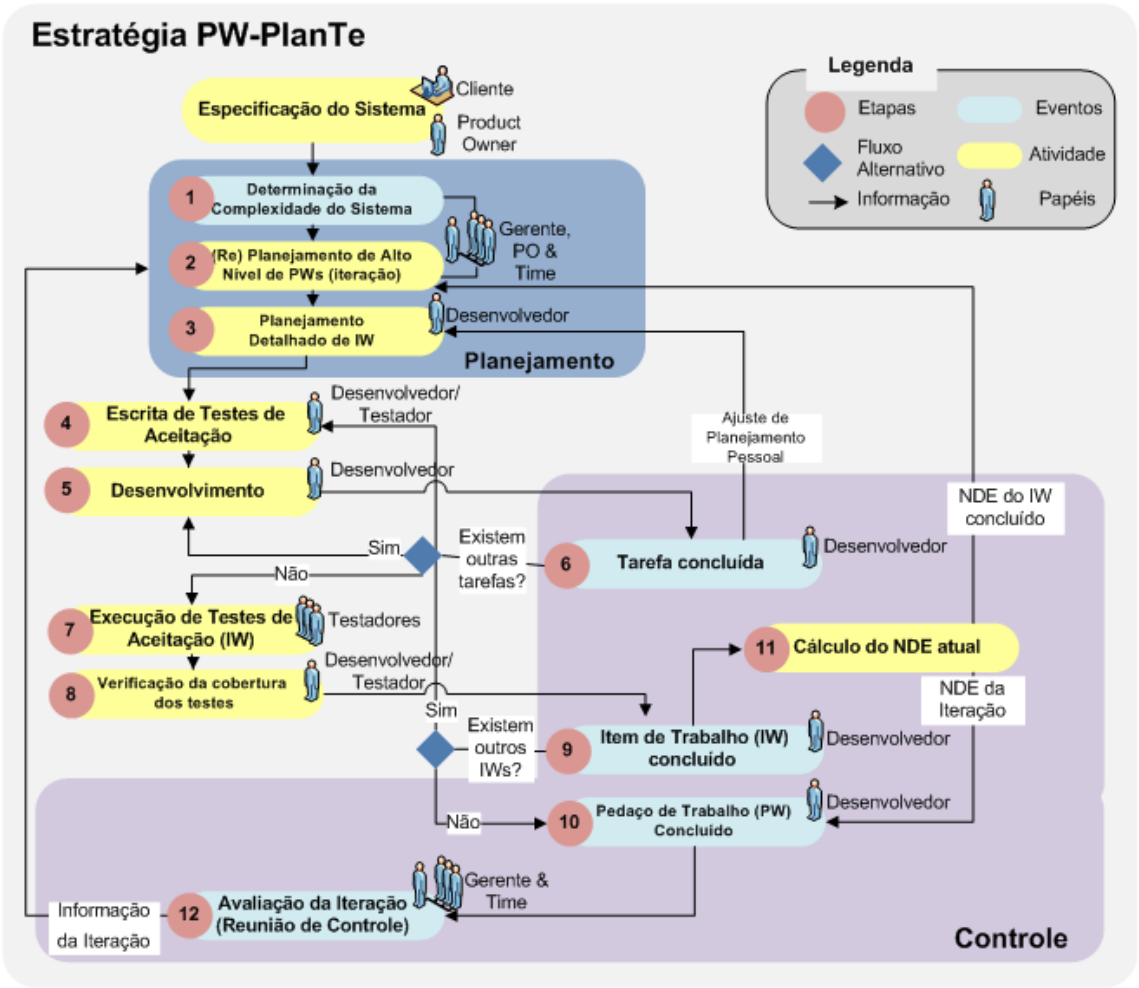


Figura 5.1 - Estratégia PW-PlanTe

Dado que a maior diferença entre a $PCU|_{PSP}$ e a PW-PlanTe corresponde à sistematização da atividade de teste, ela será explicada com mais detalhes em seguida.

Na Figura 5.2, é possível observar a ordem em que as atividades de teste presentes na PW-PlanTe devem ser realizadas. As etapas descritas na Figura 5.2 estão encapsuladas na estratégia PW-PlanTe, sendo que a Etapa A corresponde à Etapa 4 da Figura 5.1, as Etapas B e C correspondem à Etapa 7, e as Etapas D e E correspondem à Etapa 8.

Para cada IW selecionado para a iteração, ainda durante a reunião de planejamento devem ser discutidos possíveis cenários de teste. Por meio das informações obtidas nessa discussão, é possível que a equipe entenda melhor o funcionamento do IW e que uma lista preliminar de idéias de teste seja elaborada de forma a orientar a escrita dos casos de teste. O processo divide-se em três blocos principais que são: o planejamento e elaboração de testes; a execução dos testes e registro de defeitos; e a análise de cobertura dos testes realizados.

Com os dados registrados na lista preliminar de idéias de testes obtida na reunião de planejamento, a Etapa A do processo de teste deve ser realizada. O responsável pelos testes, baseando-se nas idéias de teste e na descrição dos IWs, deve planejar os casos de teste que devem ser realizados para este IW e escrever testes de aceitação para cada um destes itens (Etapa A). Ao finalizar um IW, os testes planejados para eles devem ser executados (Etapa B) de forma manual ou automatizada, e caso sejam descobertos defeitos eles devem ser registrados para que sejam corrigidos (Etapa C). Após a realização dos testes a cobertura deve ser obtida com o uso de uma ferramenta de cobertura (Etapa D), de maneira a verificar se o esforço de execução empregado foi suficiente para cobrir as funcionalidades desenvolvidas. Por fim os relatórios de evidência de cobertura devem ser gerados e armazenados de forma que a cobertura dos próximos IWs possa ser somada à realizada anteriormente (Etapa E). Um IW deverá ser considerado concluído quando, além de desenvolvido e testado, o PO tenha avaliado o item desenvolvido verificando se ele corresponde às necessidades do cliente.

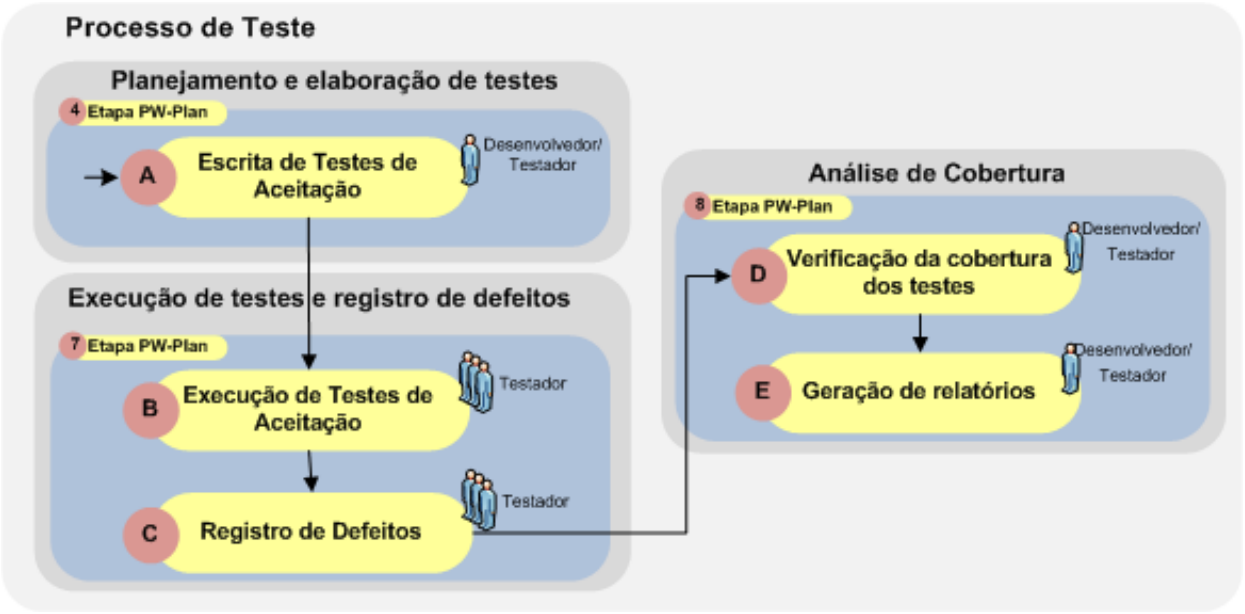


Figura 5.2 – Processo de teste existente na Estratégia PW-PlanTe

A seguir, segue o detalhamento das atividades realizadas em cada uma das etapas da estratégia PW-PlanTe, apresentada na Figura 5.1:

Etapa 1 – Determinação da complexidade do sistema: a partir da especificação do sistema, que pode ser representada por casos de uso, histórias do usuário, ou outra forma de representação, o Gerente, o *Product Owner* e o Time (desenvolvedores e testadores) discutem a complexidade do trabalho a ser desenvolvido como um todo, de acordo com o escopo do sistema que foi caracterizado pelo cliente. Essa complexidade, diferentemente da estratégia PCU|$_{PSP}$, que só permitia o uso do PCU, é caracterizada por uma técnica compatível com a representação que está sendo usada. Se a especificação for Casos de Uso, deve ser usada a técnica PCU; se forem Histórias do Usuário, deve ser usado Pontos por História e assim por diante. Caso a contagem seja realizada usando Pontos por História, a complexidade será definida com o uso de alguma técnica como, por exemplo, Planning Poker (COHN, 2005) em conjunto com a seqüência de Fibonacci. Nesse momento, é possível fazer uma estimativa da complexidade do sistema de acordo com o escopo fornecido pelo cliente, a qual é obtida por meio da soma dos pontos encontrados para a realização de cada item. Esse termo complexidade é utilizado por alguns agilistas como a representação do tamanho do sistema a ser desenvolvido. É importante ressaltar também que o valor de pontos atribuído a cada item caracteriza a complexidade com relação às atividades de

desenvolvimento e de teste. A partir desse valor, é também possível determinar o custo e o tempo de desenvolvimento multiplicando-se o valor de pontos encontrado (PCU, Pontos por História, etc.) pelo Nível de Esforço (NDE). O valor do NDE pode ser obtido da base histórica da empresa ou, caso não exista base histórica, deverá ser realizada uma estimativa inicial baseada na experiência do Time.

Etapa 2 - Planejamento de alto nível de PW (iteração): com base na especificação da Etapa 1, define-se o PW a ser realizado na iteração. Essa distribuição é realizada considerando a carga horária de desenvolvimento da iteração. Quando se tratar da primeira iteração, o NDE utilizado deve ser baseado na experiência do gerente e dos desenvolvedores ou obtido da base de dados históricos da empresa. Quando se tratar das demais iterações, o NDE utilizado corresponde ao NDE calculado na Etapa 11. Além disso, nessas outras iterações, o escopo do sistema pode sofrer alterações devido a requisitos ocultos, requisitos que devem ser eliminados ou novos requisitos, uma vez que uma das principais características dos métodos ágeis é que a equipe deve estar sempre pronta para absorver alterações. Essas mudanças podem impactar o planejamento inicial realizado na Etapa 1. Ainda durante a realização dessa etapa, com o acréscimo das atividades de teste à $PCU|_{PSP}$, a equipe discute também quais situações são candidatas a testar o IW selecionado para a iteração. O testador deve nesse momento levantar uma lista das idéias de teste discutidas que serão utilizadas para definição dos casos de teste realizado na Etapa 4 da estratégia, que pode ser visualizada também na Etapa A do processo de teste apresentado na Figura 5.2.

Etapa 3 – Planejamento detalhado de IW: cada membro do Time é responsável por dividir os IWs da iteração, que lhes foram atribuídos, em Tarefas, da maneira que julgar mais apropriada. Para cada Tarefa deve ser realizada uma estimativa em horas com base no conhecimento adquirido sobre o seu trabalho por meio do PSP. Para tanto, cada membro do Time, pode analisar a sua base de dados histórica e verificar o tempo médio gasto para realização de determinada Tarefa. Caso não existam dados históricos registrados, deve ser realizada uma estimativa inicial baseada no conhecimento adquirido e, após a conclusão da Tarefa, verificar a relação entre o tempo estimado e realmente gasto, de maneira a obter um valor mais condizente com a sua produtividade atual. De acordo com o PSP o tempo a ser registrado deve ser somente aquele realmente gasto com o desenvolvimento das Tarefas, sem considerar o tempo gasto com interrupções. Segundo Sanchez. (2008)

em um dia de oito horas, somente cinco horas são realmente consumidas no desenvolvimento, sendo que as três horas restantes são gastas com questões adjacentes.

Etapa 4 – Escrita de Testes de Aceitação: ao contrário da $PCU|_{PSP}$ que não possuía atividades formalizadas de teste, devem ser planejados e criados casos de teste para cada IW antes do seu desenvolvimento. Caso a equipe não possua testadores, os desenvolvedores devem ser responsáveis por essa atividade. Esses cenários de teste devem ser baseados no entendimento obtido durante a reunião de planejamento, na lista de idéias de teste e na descrição do IW fornecida pelo PO ou cliente. O formato desses casos de teste deve ser o mais claro possível, permitindo qualquer outro membro da equipe possa visualizar, entender e executar este teste. Os testes levantados para um IW serão executados após a conclusão do desenvolvimento desse IW, de forma manual ou automatizada.

Etapa 5 – Desenvolvimento: com base no planejamento detalhado dos IWs, realizado na Etapa 3, o desenvolvedor realiza o desenvolvimento desses IWs implementando todas as Tarefas que ele identificou para comporem os IWs em questão. Além da descrição de cada IW os desenvolvedores tem por opção verificar os cenários de teste escritos para cada IW, de forma a orientar melhor o seu desenvolvimento ou mesmo para contribuir na criação de outros cenários que sejam necessários. Durante a realização de suas atividades, o desenvolvedor deve seguir o seu processo pessoal de desenvolvimento e controlar o tempo das atividades que realiza.

Etapa 6 – Tarefa concluída: quando uma tarefa é concluída deve ser realizada uma atividade prevista no PSP denominada *post-mortem*, no que diz respeito à comparação do tempo estimado com o tempo realmente gasto. Uma maneira prática de fazer isso é obtida com o uso da *Process Dashboard*, que é o que acontece na empresa que foi parceira neste trabalho. No entanto, como já foi dito anteriormente, ainda que a empresa não use a ferramenta, a etapa pode ser realizada da mesma maneira. Dessa forma, cada membro do Time realiza o ajuste do seu planejamento pessoal, re-estimando o tempo de realização das próximas tarefas (Etapa 3) com valores mais apurados. Se houver mais Tarefas a serem realizadas, o desenvolvimento continua até que as Tarefas estabelecidas para um IW sejam concluídas.

Etapa 7 – Execução de testes de aceitação (IW): ao concluir um IW os testes de aceitação definidos na Etapa 4 devem ser realizados pelos responsáveis pelos testes. Os testes manuais são realizados por membros da equipe alocados para executar os casos de teste e alguns desses testes podem ser automatizados por meio de alguma ferramenta de automatização como, por exemplo, a Selenium. Recomenda-se que os testes sejam realizados numa versão do sistema preparada para que sejam capturados os dados de cobertura da execução desses testes. Caso sejam encontrados defeitos, eles devem ser registrados e encaminhados à equipe de desenvolvimento para correção. Esta etapa pode ser também visualizada dividida nas Etapas B e C do processo de teste exibido na Figura 5.2.

Etapa 8 – Verificação da cobertura dos testes: após a execução dos testes, os dados de cobertura devem ser analisados verificando se as instruções correspondentes às funcionalidades testadas foram executadas. A cobertura pode ser obtida por meio de ferramentas como a Emma, Cobertura, Code Coverage, dentre outras, como foi apresentado no Capítulo 3. No processo de teste exibido na Figura 5.2 esta etapa está dividida nas Etapas D e E, sendo que na Etapa E os relatórios de cobertura da aplicação dos testes devem ser gerados e armazenados em um repositório. A geração desses relatórios permite que os dados de cobertura da iteração sejam mesclados com os dados de cobertura das iterações que já foram realizadas e dessa forma seja possível obter a cobertura total dos testes realizados no sistema.

Etapa 9 - Item de Trabalho concluído: após a realização dos testes, da verificação de cobertura e do registro de defeitos (Etapas 7 e 8), o IW pode ser definido como concluído e a realização da Etapa 11 deve ser disparada. Dessa forma, um novo valor de NDE deve ser calculado, de maneira que seja possível saber se, com esta produtividade será possível realizar todos os IWs elencados para a iteração ou, caso a produtividade tenha sido baixa, se será necessário aumentar a produtividade da equipe. Este IW desenvolvido deve ser apresentado ao PO para que seja avaliado se o IW corresponde as expectativas definidas pelo cliente. Caso existam mudanças a serem feitas ou defeitos no IW desenvolvido o PO deve realizar o registro dessa requisição de mudança ou defeito e encaminhar para a equipe de desenvolvimento. Se ainda existirem IWs a serem desenvolvidos nessa iteração, o próximo IW será selecionado para realização, e o fluxo de execução retorna para a Etapa 4.

Etapa 10 – Pedaço de Trabalho concluído – Fim de iteração: quando todos os IWs elencados para a iteração são concluídos, a Etapa 11 deverá ser novamente realizada e o valor de NDE encontrado deve ser utilizado na Avaliação da Iteração, que será realizada durante a Reunião de Controle (Etapa 12) e deve ser também passado como retorno para o bloco de planejamento.

Etapa 11 – Cálculo do NDE atual: ao concluir um IW ou PW (iteração) o NDE deve ser calculado e utilizado no planejamento dos próximos IWs ou na próxima iteração. O NDE representa a relação entre o tempo efetivamente gasto para concluir um trabalho (IW ou PW) e o tamanho do trabalho realizado, ou seja, os pontos relativos ao trabalho desenvolvido, que foram calculados pela métrica adotada para fazer a reunião de planejamento.

Etapa 12 – Reunião de Controle: ao finalizar uma iteração, é promovida uma reunião de Avaliação da Iteração com o Gerente e o Time. Nessa reunião são discutidas as lições aprendidas e os problemas que ocorreram. Além disso, havendo grande diferença entre o NDE inicial e final da iteração, deve ser avaliado se algum fator externo está comprometendo o desempenho da equipe. É importante que nesse momento o Gerente auxilie o Time a refletir sobre os erros e acertos durante a iteração e a ajustar os fatores externos. Uma vez que a complexidade do trabalho a ser desenvolvido na iteração foi discutida em equipe (Etapa 2), durante essa reunião, caso tenham existido erros de estimativa, é necessário discutir, também em equipe, o porque desses erros terem acontecido. Podem ter ocorrido problemas tais como falta de conhecimento na linguagem de programação escolhida, demora na preparação do ambiente de teste ou de desenvolvimento, dentre outros.

Em resumo, por ser baseada em iterações e no controle constante das atividades realizadas por cada membro da equipe de desenvolvimento, a PW-PlanTe pode ser aplicada, com facilidade, no contexto dos métodos ágeis, em particular, o Scrum. Com as modificações aplicadas na $PCU|_{PSP}$, ela se tornou uma estratégia genérica, que permite a aplicação de diferentes técnicas para realizar a estimativa do sistema (SANDE *et al*, 2010). Além disso, a PW-PlanTe passou a dar suporte à atividade de teste, permitindo que a empresa aplique atividades de garantia de qualidade que não apenas atividades relacionadas ao planejamento do software.

5.3 Implantação da Estratégia PW-PlanTe com base no uso de softwares livres

Criar uma abordagem prática de melhoria do planejamento e da qualidade do software desenvolvido, no contexto de empresas de pequeno porte, deve levar em consideração atividades e ferramentas que causem pouco impacto na rotina de trabalho. As ferramentas devem ser, de preferência, livres de forma a causar menos impacto no orçamento das empresas, e a sua aplicação deve servir para tornar a atividade de planejamento e garantia da qualidade mais formalizada.

Entretanto, somente definir atividades e selecionar ferramentas, inserindo-as no ambiente de desenvolvimento, não é suficiente. É necessário definir um processo prático para realizar essas atividades utilizando o que as ferramentas têm de melhor.

Nas próximas subseções serão apresentados os passos de melhoria da estratégia $PCU|_{PSP}$ que permitiram a definição da PW-PlanTe e o conjunto de ferramentas livres selecionadas para apoiar a sua aplicação. Cada abordagem apresenta o histórico de uso de diferentes ferramentas livres utilizadas para dar suporte à PW-PlanTe e que permitiram também elaborar o processo de teste estabelecido apresentado na subseção 5.3.3.

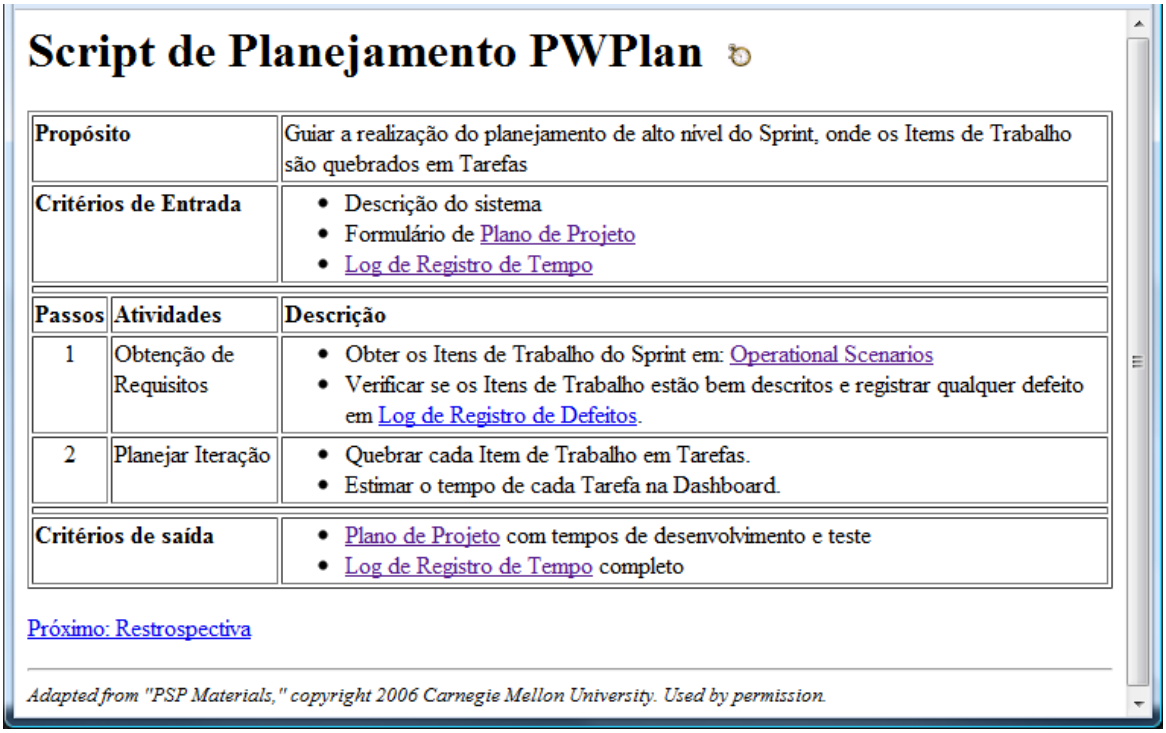
5.3.1 Passo 1 de melhoria – Acréscimo de atividades de teste com o uso da *Process Dashboard*

A primeira abordagem utilizada para melhoria e definição da estratégia PW-PlanTe, foi de dar continuidade ao uso da ferramenta *Process Dashboard* (DASHBOARD, 2010), já utilizada com a estratégia $PCU|_{PSP}$, inserindo atividades de teste que pudessem estar em conformidade com a estratégia proposta. Dessa forma, foram realizadas modificações na ferramenta *Process Dashboard* que passou a apoiar não somente o planejamento como também o processo de testes.

A ferramenta *Process Dashboard* foi desenvolvida em Java e elaborada de forma a permitir a criação de processos customizados. Os scripts e formulários do PSP não foram inseridos no código da ferramenta, em vez disso, eles foram criados separadamente e são facilmente carregados por meio de simples arquivos em texto

e HTML. Assim, foi possível criar *templates* e *scripts* que atenderam a necessidade da estratégia associando-a a dinâmica da ferramenta.

Durante a customização da *Dashboard*, foram criados *scripts* de processo que possibilitaram a definição das fases de planejamento dos IWs, desenvolvimento, testes e controle propostos pela estratégia PW-PlanTe. Na Figura 5.3 é possível visualizar o script de planejamento, que representa as etapas 1 a 3 da estratégia. Esse script foi desenvolvido baseado nos scripts do PSP disponibilizados com a Process Dashboard.



Script de Planejamento PWPlan

Propósito	Guiar a realização do planejamento de alto nível do Sprint, onde os Items de Trabalho são quebrados em Tarefas
Critérios de Entrada	• Descrição do sistema • Formulário de Plano de Projeto • Log de Registro de Tempo

Passos	**Atividades**	**Descrição**
1	Obtenção de Requisitos	• Obter os Itens de Trabalho do Sprint em: Operational Scenarios • Verificar se os Itens de Trabalho estão bem descritos e registrar qualquer defeito em Log de Registro de Defeitos.
2	Planejar Iteração	• Quebrar cada Item de Trabalho em Tarefas. • Estimar o tempo de cada Tarefa na Dashboard.

Critérios de saída	• Plano de Projeto com tempos de desenvolvimento e teste • Log de Registro de Tempo completo

Próximo: Restrospectiva

Adapted from "PSP Materials," copyright 2006 Carnegie Mellon University. Used by permission.

Figura 5.3 – Script de planejamento do template PW-Plan Sprint

Nos scripts foram descritos o propósito de cada uma das atividades previstas nos templates, os critérios de entrada, passos e critérios de saída. Alguns dos dados exibidos no *script* são fornecidos pela própria ferramenta como “Log de Registro de Tempo” e “Log de Registro de Defeitos” que também podem ser visualizados na Figura 5.3.

Dois templates foram criados, o *PW-Plan Sprint* e o *PW-Plan*. No *PW-Plan Sprint* foram definidas as reuniões de planejamento (*Planning*) e de controle (*Retrospective*) definidas na estratégia e no PW-Plan foram definidas as atividades de codificação, teste e avaliação (*Code, Test, Postmortem*), como pode ser visualizado nas Figura 5.4 e Figura 5.5.

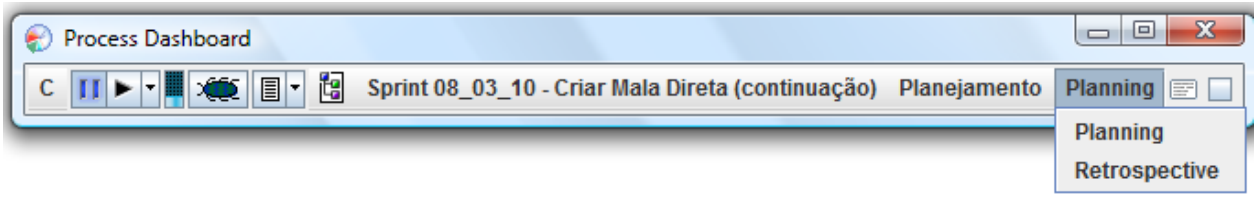


Figura 5.4 – Atividades previstas no template PW-Plan Sprint

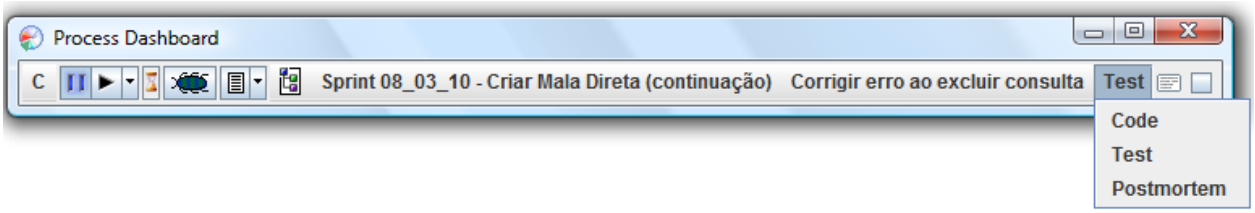


Figura 5.5 – Atividades previstas no template PW-Plan

Os templates foram criados por meio da definição de arquivos XML contendo as fases que deveriam existir em cada *template* e os *scripts* relacionados a cada fase. Esse XML foi criado de acordo com o formato disponibilizado pela ferramenta Dashboard e o código relacionado ao primeiro template pode ser visualizado na Figura 5.6:

```
<?xml version='1.0'?>

<dashboard-process-template>
   <template name="PW-PlanSprint" defectLog="true" C
dataFile="scripts/dataFile.txt" imaginaryUnless="pspProc">
 <html ID="sum" title="Project Plan Summary" href="dash/
summary.shtm"/>
 <html ID="plan" title="Script de Planejamento" inPackage="pspProc"
B href="scripts/planningSprint.htm"/>
 <html ID="pm" title="Script de Avaliacao" href="scripts/
retrospective.htm"/>

   A <phase name="Planning" htmlID="plan"    type="plan"/>
     <phase name="Retrospective" htmlID="pm" type="pm"/>
   </template>
</dashboard-process-template>
```

Figura 5.6 – XML do template PW-Plan Sprint

No *template* foi possível determinar: as etapas que deveriam ser exibidas para a coleta de tempo, que no exemplo mostrado seriam o *Planning* e o *Restrospective* (Figura 5.6 – A); os *scripts* relacionados a cada etapa (Script de

Planejamento e Script de Avaliacao) (Figura 5.6 – B); e também a opção de coletar defeitos habilitada pela definição da tag *defectLog* igual a *true* (Figura 5.6 – C).

Com os templates criados o planejamento pode ser realizado na ferramenta *Process Dashboard,* o tempo gasto com atividades de teste passou a ser incluído nesse planejamento e os defeitos puderam também ser registrados. A responsabilidade de testar o sistema, nessa primeira abordagem, recaía sobre o desenvolvedor, que era responsável também por criar os cenários de teste e realizar o registro dos defeitos. Os cenários de teste eram registrados na *Dashboard* por meio do documento *Operational Scenario Template* que continha campos para registro dos dados do sistema e dos cenários de teste, como pode ser observado na Figura 5.7.

Operational Scenarios

Name Renat(Date 19/03/2010
Project/Task /Sprint 22_03_10 - Implantar Mala Direta/Planejamento
User Administrador / Root

Construct operational scenarios to cover the normal and abnormal program uses, including user errors.

OP1 Administrador		Objetivo do usuário:	Como administrador eu quero buscar todos os contribuintes que fizeram alguma contribuição nas consultas do órgão e enviar e-mail pra esses contribuintes
Informações	Dados de entrada: grupo a enviar e-mails Campos Obrigatórios: ter um grupo selecionado (todas as consultas cadastradas pra um administrador) Restrições: não é possível selecionar mais de uma consulta por vez Funcionamento: São listadas as consultas por órgão e o administrador deve selecionar a opção enviar e-mail para todos. Pré-requisito: Estar logado como administrador		
Source	Step	Action	Comments
	1	Tendo a opção "enviar para todos" selecionada, será mostrada interface para envio do e-mail para todos os contribuintes das consultas daquele órgão e o e-mail será enviado com sucesso	Sucesso
	2	Tendo a opção "enviar para todos" selecionada, será mostrada interface para envio do e-mail para todos os contribuintes das consultas	Falha (para realizar esse teste, é necessário falhar o servidor smtp antes do primeiro e-mail ser enviado)

Figura 5.7 – Cenários de teste registrados no *Operational Scenarios Template*

Os defeitos encontrados durante a execução dos casos de teste eram registrados na Dashboard por meio da tela de "Dialogo de Defeito", exibida na Figura 5.8. Sem a definição dos *templates* e a habilitação da coleta de defeitos o registro de

defeito na *Dashboard* se tornava impossível visto que a ferramenta só permite o acesso a essa funcionalidade caso um *template* do PSP ou um *template* customizado estivesse sendo utilizado.

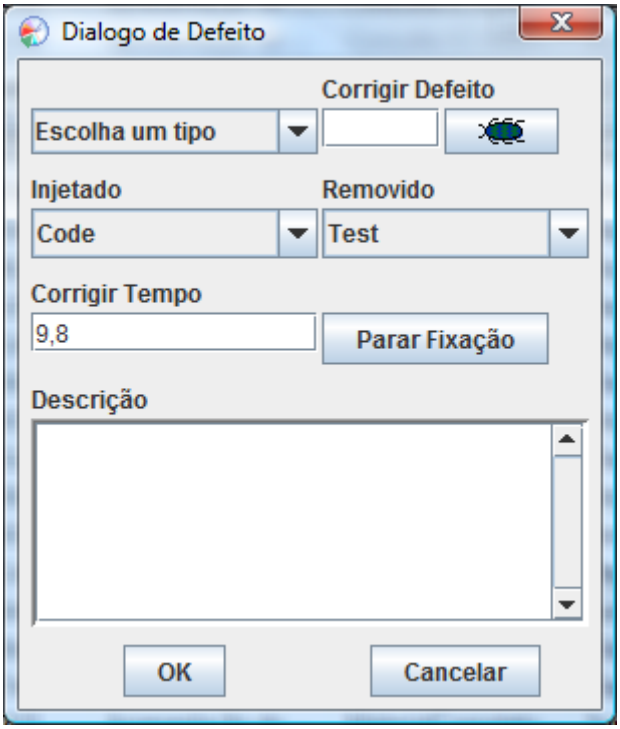


Figura 5.8 – Tela de registro de defeitos da ferramenta Process Dashboard

A customização da Dashboard para as atividades de planejamento e realização dos testes mostrou ser viável de ser aplicada e também adequada a necessidade da equipe. Com o passar do tempo, percebeu-se, no entanto a necessidade de tornar as informações de planejamento, dos testes realizados e principalmente dos defeitos encontrados, acessível a todos os participantes da equipe em qualquer momento. Tais necessidades não podiam ser solucionadas pelo uso da *Process Dashboard* que é desktop, precisa ser instalada na máquina de cada participante do time. As informações disponibilizadas pela Dashboard estão disponíveis para cada desenvolvedor, podendo ser consolidadas sob a forma de relatórios. Entretanto para ter acesso a essas informações é necessário estar conectado à máquina onde as informações foram registradas.

Devido a estas necessidades, optou-se por selecionar ferramentas que permitissem acessar os dados do projeto em qualquer instante e de qualquer lugar. Essas ferramentas deveriam funcionar de preferência em plataforma web, o que

resultou na execução do segundo passo de melhoria apresentado na subseção seguinte.

5.3.2 Passo 2 de melhoria – Gerenciamento de atividades de teste por meio de ferramentas Web

Nessa abordagem, a ferramenta Process Dashboard continuou a ser utilizada para realizar o planejamento, entretanto o registro dos casos de teste e dos defeitos encontrados passou a ser realizado nas ferramentas TestLink e Mantis, respectivamente.

Os testes passaram a ser descritos na TestLink e a execução dos testes foi realizada parte de forma manual e parte automatizada utilizando a ferramenta Selenium IDE. A cobertura dos testes foi verificada utilizando a ferramenta Cobertura.

A escolha das ferramentas TestLink e Mantis baseou-se principalmente na possibilidade do uso por meio da internet, além das facilidades de integração e de instalação bem como da grande quantidade de grupos de discussão encontrados na internet que podem auxiliar na resolução de problemas.

A ferramenta Selenium (2010) por sua vez foi escolhida por possuir uma curva de aprendizado bem menor quando comparada a outras ferramentas de automatização e também em decorrência do domínio de aplicação das atividades de teste na empresa parceira deste trabalho ser de teste em sistemas web, apesar da empresa desenvolver também sistemas desktop. A aplicação da Selenium ocorreu juntamente com a ferramenta de cobertura Cobertura que instrumenta o código fonte gerado permitindo que após a execução dos testes se obtenha dados relativos à quantidade de linhas executadas pela aplicação dos testes.

Os *scripts* gravados na Selenium eram desenvolvidos baseados nos cenários discutidos na reunião de planejamento e organizados em suítes de teste. Para cada cenário de teste levantado um script de teste era criado, como pode ser visualizado na Figura 5.9.

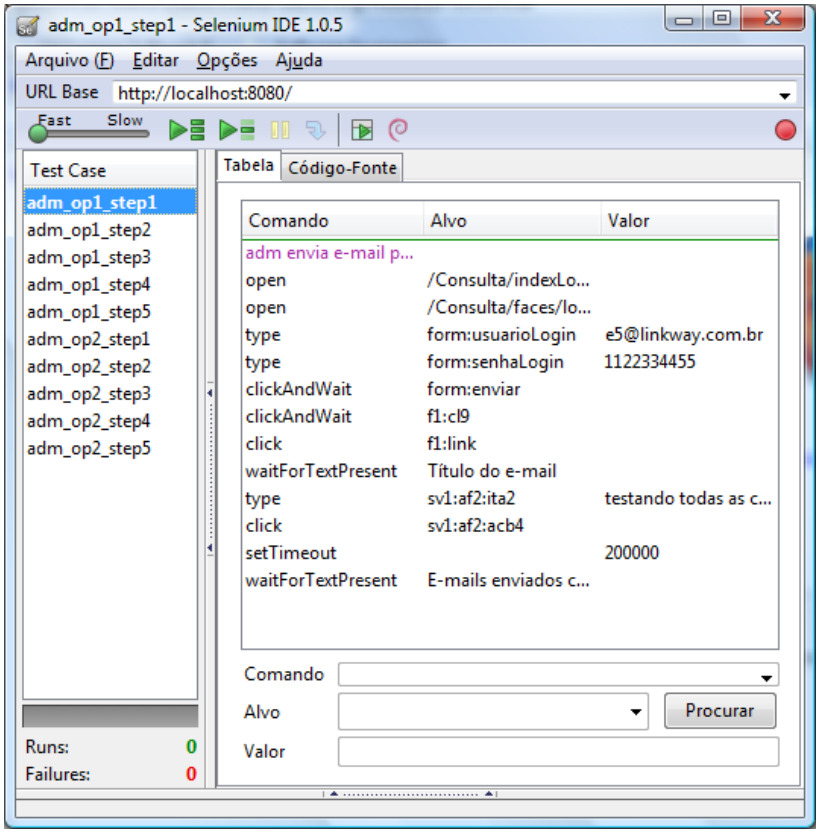


Figura 5.9 – Scripts de teste registrados na Selenium IDE

Após iniciar o uso dessas ferramentas em conjunto, percebeu-se que apesar de efetivas sobrecarregavam o time com a constante troca de contexto para administrar informações em diferentes ferramentas. Além disso, com exceção da TestLink e Mantis, as ferramentas não possuíam conexão entre si, tornando a rastreabilidade de itens de uma para outra difícil de realizar.

Essa nova necessidade fez com se optasse por usar uma ferramenta que possuísse maior suporte na realização das atividades tanto de planejamento quanto de teste e que fosse acessível de qualquer lugar, o que resultou no passo de melhoria seguinte em que a ferramenta FireScrum foi selecionada. A FireScrum reúne um conjunto de aplicações integradas para suportar equipes que utilizam o Scrum como base para o desenvolvimento de seus projetos possui além do módulo de planejamento, os módulos de gerenciamento de testes e controle de defeitos. Além disso, a ferramenta possibilita que uma rastreabilidade entre itens de trabalho, testes e defeitos possa ser realizada.

5.3.3 Passo 3 de melhoria – Definição do processo de teste com o apoio de ferramentas livres

Essa abordagem, obtida por meio dos aprendizados com as abordagens anteriores, é o cenário final de definição da estratégia. Ela prevê uma união dos passos de melhoria 1 e 2, substituindo as ferramentas TestLink e Mantis pela FireScrum apresentada no capítulo 4.

Apesar de suprir algumas necessidades da estratégia PW-PlanTe, a ferramenta FireScrum não possui ainda em suas funcionalidades o controle do tempo realmente gasto para realização das atividades e também a possibilidade de automatização de testes e verificação de cobertura. Dessa forma, para suportar a estratégia PW-PlanTe no que diz respeito ao planejamento e teste foram utilizadas as seguintes ferramentas: Process Dashboard para a realização da estimativa e controle do tempo; FireScrum para a realização do planejamento do trabalho, registro de testes e registro de defeitos; Selenium IDE para realização dos testes automatizados; Cobertura para verificação da cobertura dos testes realizados.

Os IWs discutidos na Reunião de Planejamento devem ser registrados na FireScrum, que permite além da criação de Itens de Trabalho (*Backlog Itens*) a customização da medida utilizada. Ao criar um produto na FireScrum é possível associar uma medida que podem ser horas, pontos, dias, etc. Na tela principal da FireScrum são exibidas duas colunas: o *Product Backlog* e o *Commited Backlog*. No *Product Backlog* devem ser registrados todos os itens levantados para o sistema e também a complexidade obtida para cada IW durante a Etapa 1 da PW-PlanTe. Ao criar um IW deve ser registrada também a sua descrição, que pode ser refinada durante as próximas reuniões de planejamento, quando este IW for selecionado para uma iteração. No *Commited Backlog* serão criadas as iterações (*Sprints*), e os IWs que forem selecionados para compor este PW devem ser arrastados da coluna *Uncommited Backlog* Items para o Sprint criado. Após a reunião, cada membro do Time deve selecionar os itens que lhe foram atribuídos e quebrar em Tarefas, estimando cada Tarefa em horas. O planejamento do trabalho pode ser visualizado na Figura 5.10.

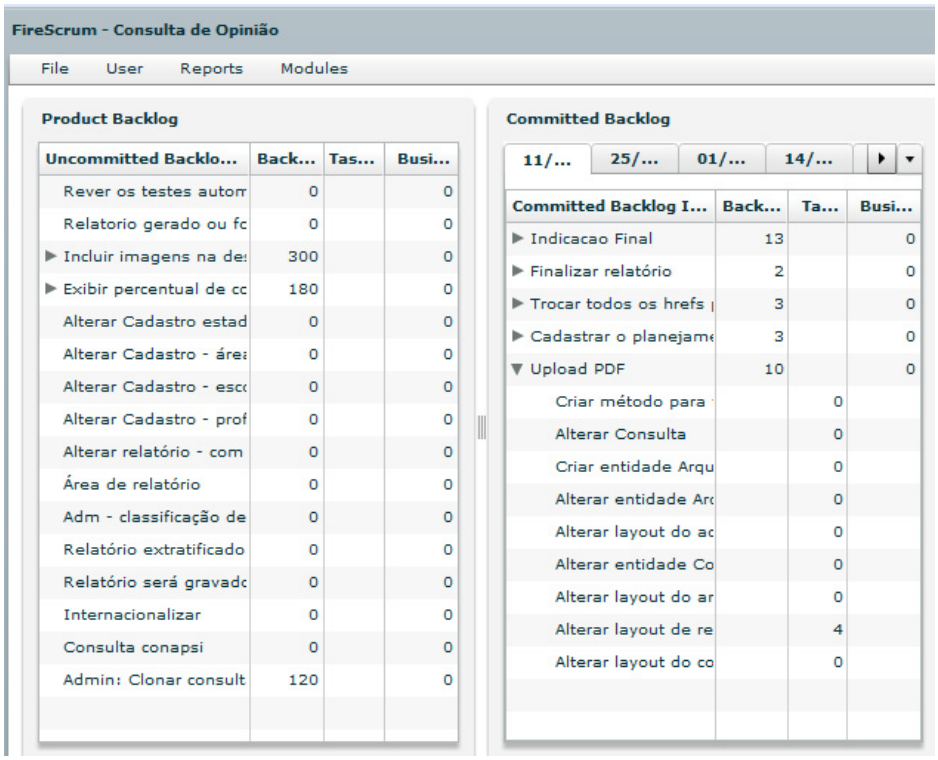


Figura 5.10 – FireScrum: *Product Backlog* e *Commited Backlog*

Para realizar o controle do tempo a mesma representação de IWs e Tarefas criada na *FireScrum* deve ser reproduzida na *Process Dashboard*. Na *Dashboard* serão registradas as mesmas estimativas de tempo associadas às Tarefas na *FireScrum* utilizando os *templates PW-Plan Sprint* e *PW-Plan* mostrados na passo de melhoria 1. Por meio da *Dashboard* o tempo será medido e controlado permitindo que possa ser efetuado o cálculo do NDE. Por meio dos módulos *Core* e *Task Board* da *FireScrum*, o trabalho será acompanhado por toda a equipe. No módulo *Task Board* as Tarefas que ainda não começaram a ser desenvolvidas devem ser mantidas na coluna *To Do*, as Tarefas em desenvolvimento devem ser mantidas na coluna *In Progress*, as Tarefas que estejam de alguma forma impedidas de serem realizadas devem ser mantidas na coluna *Impeded*, e por fim, as Tarefas concluídas devem ser transferidas para a coluna *Done*.

O processo de teste, apresentado na Seção 5.2, pode ser aplicado de duas formas: com o suporte das ferramentas FireScrum e Cobertura, sendo que nesse caso o processo foi denominado Processo Manual de Teste, pois a execução dos testes é realizada de forma manual; e com o suporte das ferramentas FireScrum, Selenium IDE e Cobertura, sendo que o processo nesse caso foi denominado

Processo Automatizado de Teste, no qual a execução dos testes pode ser automatizada com o uso da ferramenta Selenium IDE.

Dessa forma, o processo manual exibido na Figura 5.11, ocorre com o apoio das ferramentas *FireScrum* e Cobertura. As etapas descritas na Figura 5.11 estão encapsuladas na estratégia PW-PlanTe, sendo que a Etapa 1 corresponde à Etapa 4 apresentada na estratégia, as Etapas 2 e 3 correspondem à Etapa 7, e as Etapas 4 e 5 correspondem à Etapa 8.

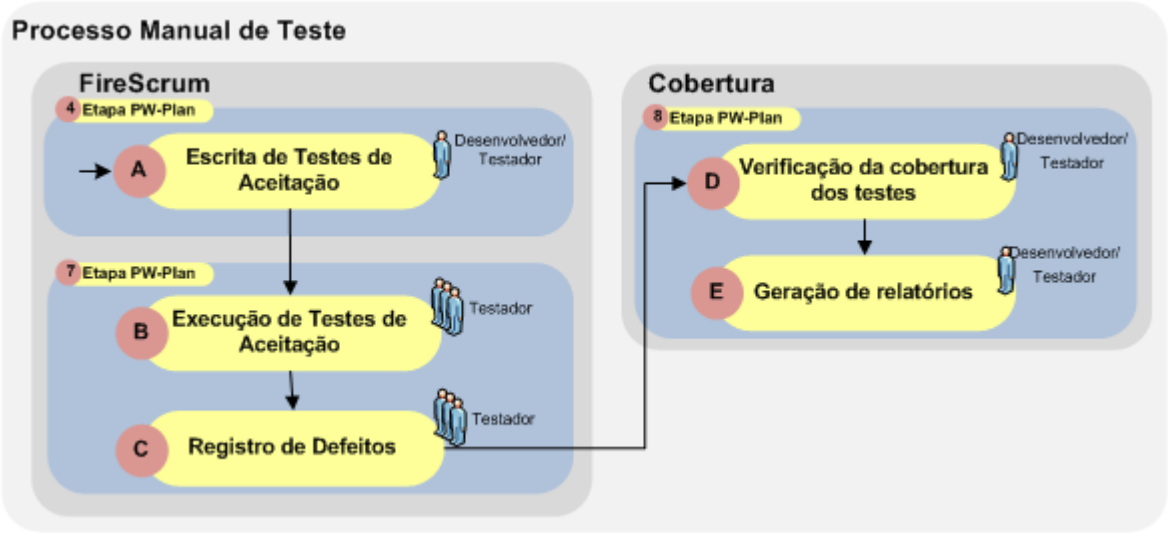


Figura 5.11 – Processo Manual de Teste

Primeiramente os casos de teste são descritos na ferramenta FireScrum (Etapa 1) e após a fase de desenvolvimento, um plano de teste é gerado para a alocação dos casos de teste que deverão ser executados. Para criar um caso de teste é necessário que já tenha sido criada uma suíte de testes.

Na Figura 5.12 é possível visualizar as telas de criação de um novo caso de teste onde para cada caso de teste devem ser preenchidos: 1- as informações gerais do teste (*General Information*); 2 - informações adicionais (*Additional Information*); e 3 - os passos de execução do caso de teste (*Steps*). Na tela de informações é obrigatória a inserção do nome do caso de teste e descrição. Entretanto outras informações importantes devem ser preenchidas como o *Backlog Item* ao qual este teste está associado, o tipo do teste (*Functional, Stress, Performance*), o tipo de execução (*Manual, Semi-Automatic, Automatic*) e a prioridade de execução do teste. Na tela de informações adicionais devem ser informados o cenário de teste, condições iniciais (preenchimento obrigatório) e notas de execução. Por fim, na tela

dos passos de execução são registrados os passos de execução do caso de teste, no formato “procedimento”-“resultado esperado”.

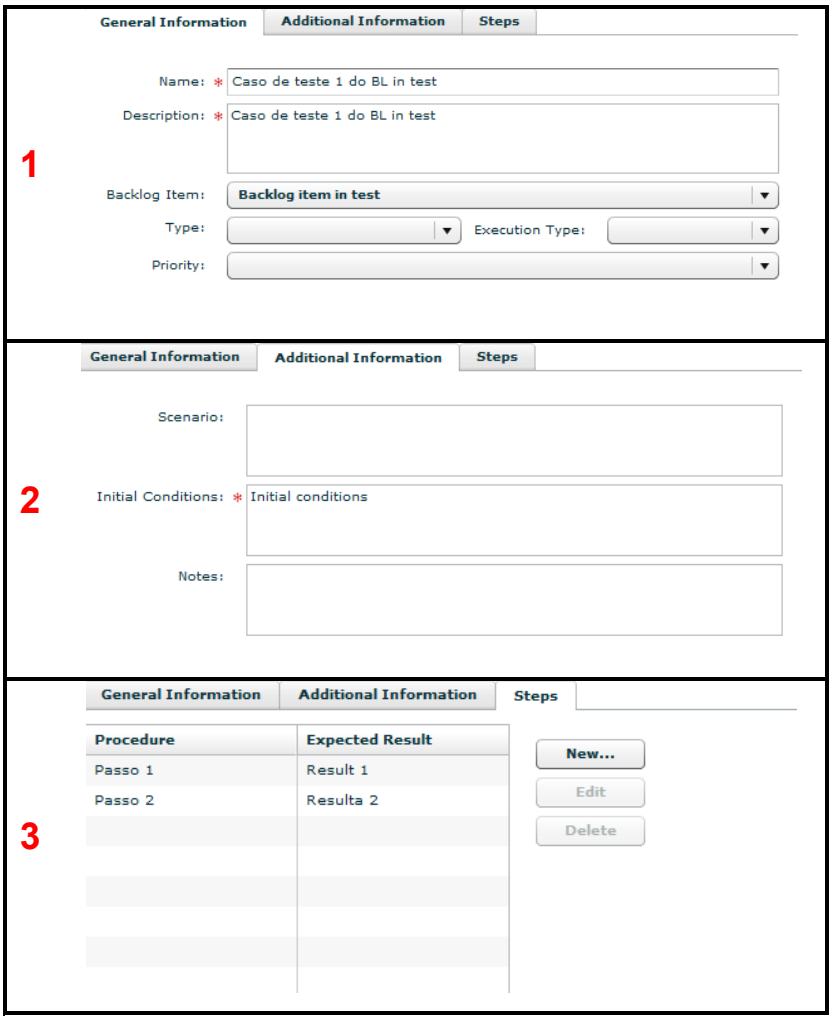


Figura 5.12 - Passos para criação de um caso de teste na FireScrum

Após a criação dos casos de teste para um determinado IW, é possível executar os casos de teste no sistema (Etapa 2) e também registrar os resultados dessa execução. Para que um caso de teste seja executado é necessário que seja criado um plano de teste e neste plano de teste um ciclo de teste. No plano de teste devem ser inseridos todos os testes que deverão ser realizados na iteração e para executar estes testes um ciclo de teste deve ser criado (Figura 5.13).

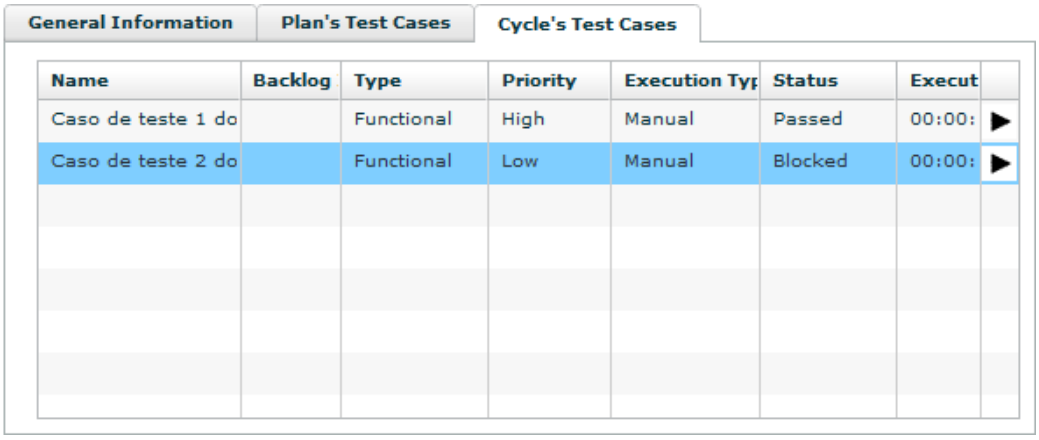


Figura 5.13 - FireScrum: Inclusão de testes no ciclo de testes

Um caso de teste pode, ao ser executado (Figura 5.14), ser marcado como *Passed*, *Failed* ou *Blocked*. Além disso, o tempo gasto para sua execução pode ser coletado bem como qualquer comentário ou nota de execução pode ser também registrado. Os dados da execução dos casos de teste ficam armazenados na ferramenta e podem ser visualizados por qualquer membro da equipe.

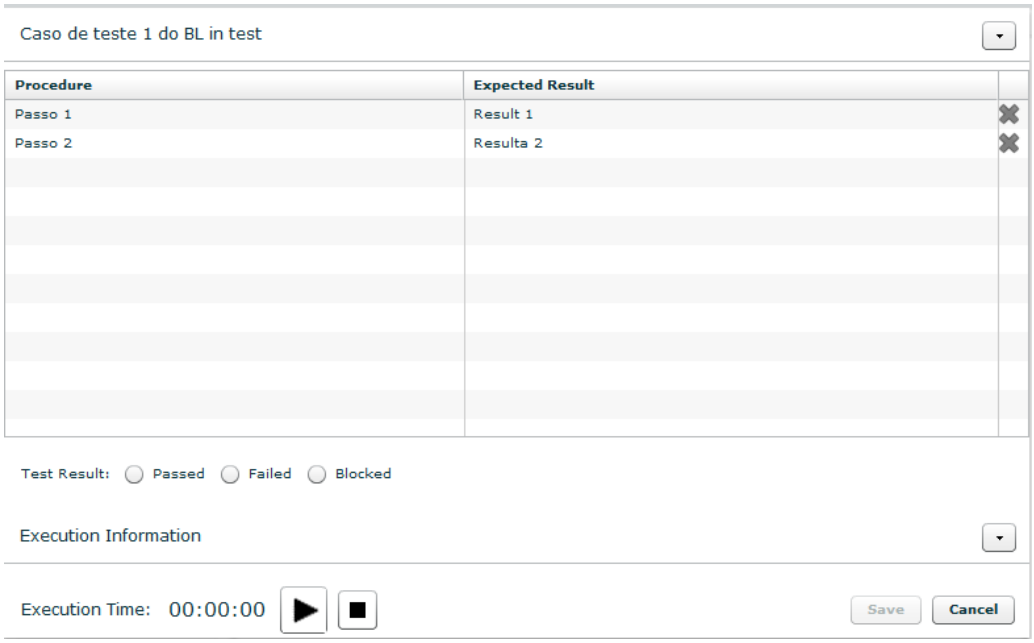


Figura 5.14 - FireScrum: Tela de execução dos testes

Após a execução dos testes, caso algum defeito seja encontrado deverá ser registrado no módulo de *Bug Tracking* da FireScrum e direcionado para o responsável por corrigi-lo (Etapa 3). Durante o registro do defeito a ferramenta

permite anexar arquivos que possam auxiliar no entendimento do problema e permite também determinar uma data final para sua correção.

Terminada as etapas de execução e registro dos defeitos, é possível verificar a cobertura dos testes realizados utilizando a ferramenta Cobertura (Etapa 4). Para tanto é necessário que antes da execução a aplicação sob teste tenha sido "preparada" adicionando a instrumentação da Cobertura ao *bytecode* que seria executado. A Cobertura, assim como apresentado no capítulo 3, insere instruções de instrumentação diretamente no código Java compilado. Ao executar a aplicação a Máquina Virtual incrementa vários contadores no momento em que encontra as instruções de instrumentação. Dessa forma é possível dizer quais instruções foram cobertas e quais não.

A ferramenta de cobertura permite também que sejam gerados relatórios que informam a percentagem de código alcançado pelos testes realizados que corresponde a quantidade de linhas executadas (Etapa 5).

O processo manual de teste pode ser incrementado pela automatização de alguns dos testes de aceitação a serem realizados. Para tanto, os testes candidatos a serem executados diversas vezes em uma mesma aplicação podem ser automatizados de forma que possam ser executados sempre que necessário. Esse processo, apresentado na Figura 5.15, é apoiado pelas ferramentas FireScrum, Selenium IDE e Cobertura. Da mesma forma que o processo manual descrito na Figura 5.11 o processo automatizado descrito na Figura 5.15 está encapsulado nos passos 4, 7 e 8. O diferencial deste processo com relação ao processo manual é a inserção de atividades realizadas com a ferramenta Selenium IDE.

A etapa 1 corresponde à etapa 4 da estratégia apresentado na Figura 5.1, as etapas 2, 3 e 4 correspondem à etapa 7, e as etapas 5 e 6 correspondem à etapa 8.

Primeiramente os casos de teste que deverão ser automatizados devem ser também descritos na ferramenta FireScrum (Etapa 1) e o seu Tipo de Execução é marcado como *Automatic*. Após a fase de desenvolvimento, são criados na Selenium IDE os scripts de teste que serão aplicados (Etapa 2). Após a criação desses scripts eles serão executados (Etapa 3) e os resultados dos testes serão marcados na ferramenta FireScrum como *Passed*, *Failed* ou *Blocked* e os defeitos registrados no *Bug Tracking* (Etapa 4).

Após a execução dos testes a cobertura é verificada por meio da ferramenta Cobertura (Etapa 5) e os relatórios gerados pela ferramenta Cobertura bem como os

resultados dos testes podem ser então analisados para verificar a efetividade dos testes realizados.

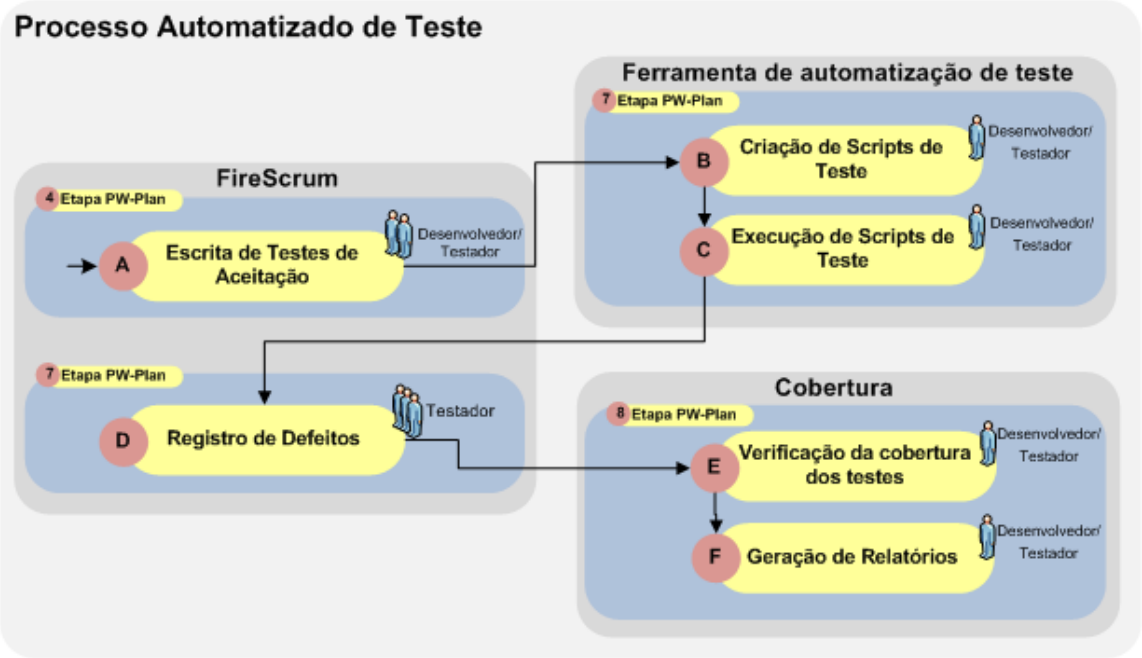


Figura 5.15 - Processo Automatizado de Teste

Caso sejam aplicados testes manuais e automatizados, os dados de cobertura obtidos devem ser mesclados gerando um relatório com a cobertura total da atividade de teste realizada. Com esses dados em mãos o Time pode tomar a decisão de aumentar o esforço dedicado aos testes de forma a cobrir partes importantes ou mesmo alcançar a quantidade total de linhas desenvolvidas.

Com a aplicação de testes funcionais e da ferramenta de cobertura é possível verificar o resultado da execução dos testes diretamente nas linhas de código, o que permite rastrear a cobertura estrutural dos testes realizados.

As atividades selecionadas buscam apoiar a aplicação de forma prática e sem um grande custo atividades de planejamento e controle bem como de VV&T, no contexto de empresas de pequeno porte. Para tanto, as ferramentas selecionadas são todas livres, não possuem custos para a sua aquisição, e buscam apoiar a aplicação das atividades de teste e também do gerenciamento ágil.

O desenvolvimento e aplicação desse processo tornam claro que é possível, mesmo com poucos recursos financeiros e humanos envolvidos, realizar atividades básicas de teste. Dessa forma, é possível auxiliar empresas de pequeno porte a sair

de um estado onde as atividades são feitas de maneira informal e sem continuidade, a um estado com processos melhor definidos e com uma maturidade maior.

Por ter sido construído baseado em observação e aplicação de atividades práticas foi possível experimentar diversas abordagens, e coletar o melhor de cada abordagem de forma a tornar o processo mais leve e utilizável possível.

5.4 Estratégia PW-PlanTe e o Scrum

A Estratégia PW-PlanTe, como já dito anteriormente, foi desenvolvida com base na Estratégia $PCU|_{PSP}$ e suas recomendações, assim como a $PCU|_{PSP}$ possuem bastante similaridades com o framework ágil Scrum. Suas recomendações se complementam visto que enquanto o Scrum define “o que deve ser feito” a estratégia define “como deve ser feito” (SANCHEZ, 2008). Dessa forma, a estratégia PW-PlanTe busca responder questões de como estimar o trabalho a ser realizado, como os itens de trabalho do *Product Backlog* devem ser inseridos no *Selected Product Backlog*, como devem ser estimadas as tarefas do Sprint, como gerenciar o desenvolvimento dos *Backlog Itens* e a velocidade da equipe, e como o *Scrum Master* avalia a evolução dos Desenvolvedores.

Os papéis do Scrum relacionam-se aos papéis da Estratégia PW-PlanTe como mostrado anteriormente sendo que o *Scrum Master* é representado pelo Gerente; a equipe de desenvolvimento corresponde ao Time; o *Product Owner* (PO) é o representante do cliente para a equipe de desenvolvimento.

O PO é o responsável pelo *Product Backlog* e deve definir quais são funcionalidades do produto de acordo com as necessidades estabelecidas pelo cliente. Ele é o representante do cliente na empresa e responde pela rentabilidade do produto, priorizando as funcionalidades de acordo com o valor de mercado e aceitando ou rejeitando os resultados do trabalho realizado pelo Time de desenvolvimento. Na estratégia PW-PlanTe o PO além de participar da reunião de planejamento ele deve ser também responsável por acompanhar o desenvolvimento do trabalho por meio da FireScrum, priorizando antes das reuniões os itens do backlog. Além disso, o PO é responsável por realizar testes informais de aceitação confirmando que o produto foi desenvolvido conforme o esperado.

O *Scrum Master*, representado na estratégia PW-PlanTe pelo gerente, é responsável por garantir que o Time de desenvolvimento se oriente pelas recomendações estabelecidas e assuma somente o trabalho que possa ser cumprido durante uma iteração. O *Scrum Master* deve ser responsável também por organizar as reuniões e remover os impedimentos que surjam no decorrer do trabalho. Na estratégia PW-PlanTe o *Scrum Master* deve ser também responsável por acompanhar por meio da FireScrum o trabalho realizado pelo Time, a resolução dos defeitos e o resultado dos testes. Por meio dos resultados obtidos por meio da Dashboard o *Scrum Master* deve acompanhar o tempo gasto com o desenvolvimento por cada membro do Time e realizar ajustes no planejamento por meio do cálculo do NDE. Com os dados obtidos por meio da ferramenta de cobertura o gerente deve acompanhar a cobertura dos testes realizados e verificar se é necessário aumentar o esforço empregado em testes.

O Time de desenvolvimento é formado pelos desenvolvedores e testadores. Os desenvolvedores são responsáveis pelo desenvolvimento dos itens elencados para uma iteração e devem quebrar e estimar o trabalho a ser desenvolvido em Tarefas que serão executadas na iteração. Os desenvolvedores são responsáveis também por apresentar os itens desenvolvidos ao PO, solicitar teste dos itens, integrar e publicar o sistema desenvolvido e analisar e corrigir os defeitos encontrados. Na FireScrum o desenvolvedor deve registrar e estimar as Tarefas, analisar e corrigir os defeitos registrados e também acompanhar o resultado dos testes realizados. Os relatórios gerados pela ferramenta de cobertura devem ser também analisados verificando se condizem com os resultados dos testes relatados na FireScrum. Por meio da Dashboard o desenvolvedor deve registrar e acompanhar o tempo gasto em suas atividades. Os testadores assim como os desenvolvedores devem acompanhar por meio da Dashboard o desenvolvimento das suas atividades, sendo que os testadores são responsáveis pela realização dos testes dos itens elencados para uma iteração. Os testadores devem criar casos de teste para cada item liberado pelo desenvolvimento para teste, registrar o resultado dos testes, registrar os defeitos encontrados, acompanhar a correção dos defeitos encontrados no sistema e gerar e acompanhar os resultados de cobertura dos testes realizados.

A estratégia PW-PlanTe contempla as atividades Planejamento 1 do *Sprint*, Planejamento 2 do *Sprint*, atividades de acompanhamento do *Sprint* e a atividade de Retrospectiva, como pode ser observado na Figura 5.16.

Os requisitos do sistema levantados pelo *Product Owner* são denominados no *Scrum* como *Product Backlog*. Com base nesse *Product Backlog* é realizada a reunião de planejamento 1 do sprint, onde o PO, o *Scrum Master* e o Time definem as prioridades do produto a ser desenvolvido. Na estratégia PW-PlanTe esta etapa é realizada por meio da primeira reunião de planejamento representada pela Etapa 1, onde os itens que compõem o *Product Backlog* são estimados quanto ao seu tamanho. Por meio dessa estimativa é possível derivar estimativas de prazo e custo iniciais. A estratégia PW-PlanTe permite que seja possível calcular o tempo despendido para desenvolver cada item do sistema baseando-se na complexidade atribuída por meio dos pontos e no NDE da equipe de desenvolvimento. Por meio da estratégia PW-PlanTe é possível selecionar somente os itens do *Product Backlog* que cabem em uma iteração ou *Sprint*.

Após a definição do *Product Backlog* as Etapas 1, 2 e 3 da estratégia que correspondem ao Planejamento 2 do *Sprint* devem ser executadas refinando a complexidade atribuída para os itens selecionados para o próximo *Sprint*. Cada membro do Time deve então quebrar os itens atribuídos em Tarefas e realizar a estimativa de tempo para cada Tarefa.

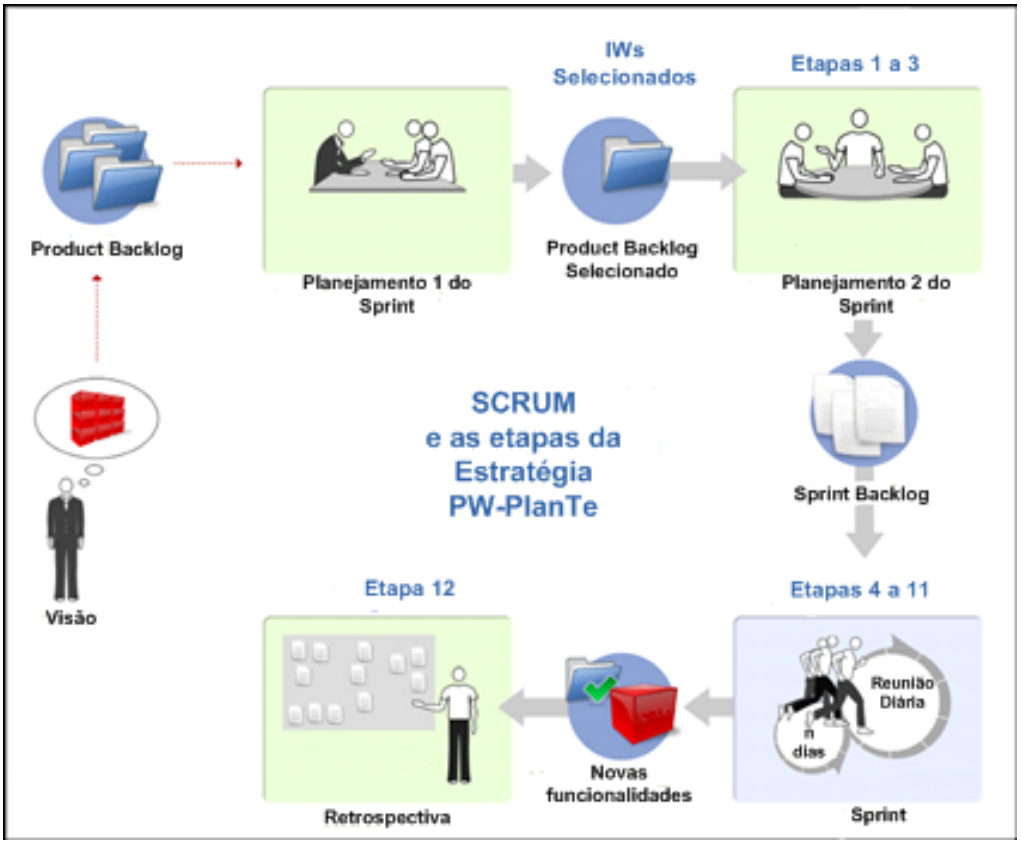


Figura 5.16 - Scrum e a Estratégia PW-PlanTe (Adaptado de (SANCHEZ, 2008))

Durante o desenvolvimento do sistema o Time deve realizar as etapas de 4 a 11 da estratégia, onde devem realizar também o controle do tempo e o cálculo do NDE. Por meio do cálculo do NDE proposto pela estratégia PW-PlanTe, as atividades do *Sprint* podem ser acompanhadas permitindo ao *Scrum Master* replanejar, caso seja necessário o trabalho que cabe nos próximos *Sprints*.

Durante a reunião de retrospectiva, correspondente a Etapa 12 da estratégia PW-PlanTe, todas as atividades e tarefas realizadas devem ser revisadas e os problemas ocorridos devem ser discutidos e, dentro do possível, solucionados.

Durante a realização do *Sprint* as atividades que tenham tido algum impedimento, devem ser levantados durante as reuniões diárias e de retrospectiva, de forma que toda a equipe possa estar a par dos problemas existentes.

Finalmente, na atividade de Retrospectiva é feita uma revisão de todas as atividades e tarefas realizadas ao longo de todo processo e deve ser analisado o que funcionou e o que precisa ser melhorado.

O *Scrum Master* atua como facilitador nas reuniões diárias previstas no método *Scrum*, e torna-se o responsável por remover quaisquer obstáculos que sejam levantados pela equipe nessas reuniões. A lista desses impedimentos é conhecida como *Impediment Backlog,* que devem ser discutidos durante as reuniões diárias e solucionados.

5.5 Considerações finais

Neste capítulo foi apresentada a estratégia PW-PlanTe, que tem por objetivo apoiar o planejamento do desenvolvimento de software permitindo que o planejamento possa ser realizado e acompanhado por meio do ajuste do Nível de Esforço da equipe. A estratégia prevê também a realização de atividades de teste definidas com o objetivo de auxiliar a pequenas empresas de desenvolvimento de software aplicar atividades básicas de teste a um baixo custo.

A definição desta estratégia foi elaborada com os aprendizados obtidos na adaptação da estratégia $PCU|_{PSP}$ e na aplicação de uma sequência de passos de melhoria e aplicação das atividades de teste. Por ter sido desenvolvida em um processo constante de observação e evolução a estratégia permitiu a proposta de

atividades que possam ser utilizados no contexto de empresas de pequeno porte mesmo com poucos recursos disponíveis.

Durante a definição desta estratégia foi possível perceber também como a pequena empresa de software é carente de orientações práticas que auxiliem a melhoria de processo. Estabelecer meios de auxiliar essas empresas a saírem de um estado caótico para um estado mais formalizado é um grande desafio a ser vencido. Nesse sentido, a estratégia PW-PlanTe permite auxiliar pequenas empresas de software no planejamento do desenvolvimento e na distribuição das tarefas a serem realizadas. Por meio da complexidade obtida é possível elaborar estimativas de tempo e esforço e, com o cálculo do NDE sendo realizado na finalização de cada item, é possível identificar os problemas e resolvê-los de forma rápida.

No próximo capítulo serão apresentados os resultados obtidos por meio da aplicação da estratégia e do processo de testes na empresa de desenvolvimento onde a estratégia foi estabelecida.

Capítulo 6

ESTUDO DE CASO

Neste capítulo são apresentados quatro estudos de caso realizados para avaliar o uso da Estratégia PW-PlanTe. Os dois primeiros estudos de caso exploraram a aplicação da estratégia para o planejamento de software enquanto que os dois últimos exploraram a qualidade de software, de forma a validar o processo de teste proposto.

6.1 Considerações Iniciais

Segundo Basili *et al* (1996), a simples proposta de métodos, técnicas, ferramentas, etc., sem mostrar a aplicação e a caracterização de como, e em quais circunstâncias a proposta pode contribuir, não colabora para o avanço da ciência, em particular, para o desenvolvimento da área de engenharia de software.

No caso deste trabalho, como a estratégia proposta – PW-PlanTe – foi definida gradativamente com base na extração, proposição e adoção de práticas em tempo real, junto aos desenvolvedores, ao mesmo tempo em que a estratégia era definida, sua aplicação prática caracterizava um estudo de caso. Assim, como dito no Capítulo 6, o desenvolvimento deste trabalho pode ser caracterizado como uma indústria como laboratório (POTTS, 1993).

Portanto, alguns estudos apresentados neste capítulo correspondem a exemplos que foram realizados durante a criação e evolução da estratégia e do processo de teste nas empresas Linkway e NBS. Outros foram aplicados posteriormente à definição da estratégia ter sido completada.

Dessa forma, neste capítulo serão apresentados os estudos realizados, que estão organizados da seguinte forma: na Seção 6.2 são caracterizadas as empresas

que participaram dos estudos e como os estudos foram realizados; na Seção 6.3 são apresentados os resultados obtidos da aplicação da estratégia com o intuito de mostrar sua contribuição para o planejamento do software; na Seção 6.4 explora-se a contribuição da estratégia quanto à atividade de teste; assim, é mostrado o resultado da aplicação do processo de teste de duas formas: em comparação ao processo informal que era adotado pela empresa e na sua aplicação em um sistema já desenvolvido, para mostrar a eficácia do processo. Por fim, na Seção 6.5 apresentam-se as considerações finais sobre os resultados apresentados neste capítulo.

6.2 Caracterização das empresas e dos estudos de caso

As empresas nas quais a estratégia e o processo foram definidos e aplicados, foram a Linkway e a NBS, localizadas na cidade de São Carlos. A Linkway é uma empresa de pequeno porte que, além de prover internet, desenvolve aplicações Web. A NBS é uma pequena empresa de software, focada no desenvolvimento de aplicativos de gestão e administração pública.

A Linkway tem uma equipe de desenvolvimento formada por um *gerente*, três consultores de vendas, dois *Web Designers,* dois Desenvolvedores Java e três atendentes de suporte, dentre os quais um atua como testador. A empresa está no mercado de desenvolvimento de software web há quinze anos, e por ser de pequeno porte, possui um ciclo de desenvolvimento com entregas mais rápidas, que depende do cumprimento dos prazos, bem como da qualidade do produto para se manter no mercado.

A NBS possui uma equipe de desenvolvimento formada por um *gerente*, um desenvolvedor, dois analistas de suporte e um consultor de vendas. A NBS assim como a Linkway, é uma empresa de pequeno porte; entretanto, o mercado no qual atua já possui requisitos mais estáveis, pois as regras de negócio da área de administração pública não mudam com a mesma intensidade de outros setores. Apesar do foco do seu desenvolvimento ser sistemas *Desktop,* a NBS passou, nos últimos anos, a desenvolver também sistemas *Web*. Para facilitar o aprendizado e troca de informações sobre esse tipo de sistema, a NBS optou por alocar a equipe

de desenvolvimento web na *Linkway*, que já possui maior experiência na área. Dessa forma, a troca de experiências entre as equipes da NBS e Linkway auxiliam que no desenvolvimento web na NBS possa ocorrer de maneira melhor estruturada.

Os resultados apresentados referem-se ao trabalho realizado durante o acompanhamento de 3 sistemas web desenvolvidos pelas empresas Linkway e NBS. A definição das atividades realizadas pode ser vista na Tabela 6.1. No sistema CondLink, desenvolvido pela Linkway, a PW-PlanTe foi aplicada com o objetivo de validar a estratégia no que diz respeito ao planejamento. No sistema Consulta Pública, desenvolvido pela NBS, a estratégia foi aplicada com o objetivo de validar a estratégia tanto com relação ao planejamento como com relação ao processo de teste criado. Já no sistema Toalhas São Carlos, desenvolvido pela Linkway, a estratégia foi aplicada em um sistema já concluído com o objetivo de também validar o processo de teste.

Tabela 6.1 – Organização dos estudos de caso

Sistema	Empresa	Planejamento	Teste	Objetivo
A. CondLink	Linkway	✓	✓	Avaliar a PW-PlanTe no contexto de planejamento e gerenciamento de software por meio do acompanhamento das iterações de desenvolvimento.
B. Consulta Pública	NBS	✓	✓	Avaliar a PW-PlanTe da seguinte forma: no contexto de planejamento e gerenciamento de software por meio do acompanhamento das iterações de desenvolvimento; no contexto de teste realizando um comparativo entre os dados de cobertura e defeitos coletados com o processo ad-hoc e com a aplicação da PW-PlanTe.
C. Toalhas São Carlos	Linkway		✓	Avaliar a PW-PlanTe no contexto de teste de software demonstrando a aplicação do processo de teste em um sistema já desenvolvido e já no ambiente de produção, de forma a verificar se seriam identificados problemas que não foram descobertos por não existir um processo definido

De forma que seja possível visualizar melhor as contribuições relacionadas ao planejamento e ao teste de software, nas próximas seções os estudos foram organizados da seguinte forma: na Seção 6.3 são apresentados os estudos de caso aplicados nos sistemas A e B, explorando a atividade de planejamento de software; e na Seção 6.4 são apresentados os estudos realizados nos sistemas B e C, explorando a atividade de teste de software. Ressaltam-se dois pontos:

(i) No sistema A, embora as atividades de teste também tenham sido realizadas, como era a primeira vez que a equipe estava aplicando o processo de teste juntamente com o uso das ferramentas, nem todas as informações foram registradas nas ferramentas. Apenas o tempo gasto na atividade de teste foi registrado, como será visto na seção seguinte. Assim, esse fato impediu que fosse feita uma avaliação do processo de teste nesse caso.

(ii) No sistema B, em que os dados de teste estão analisados na subseção 6.4.1, as atividades de teste foram aplicadas para avaliar o processo que estava ainda sendo definido. Essas atividades foram aplicadas com a supervisão da autora e anotadas conforme instruído pela estratégia. No entanto, quando a equipe passou a aplicar as atividades de planejamento nesse sistema, as atividades de teste foram negligenciadas pelo fato da equipe ainda não estar acostumada com o novo processo que estava em fase de implantação. Esse fato salienta a dificuldade tão mencionada na literatura, referente à mudança cultural. Sabe-se que esse é um dos principais fatores que impedem, muitas vezes, a implantação de um modelo de melhoria de processo em uma empresa.

Para ficar mais claro como tudo aconteceu, de forma cronológica, esses fatos estão representados na Figura 6.1.

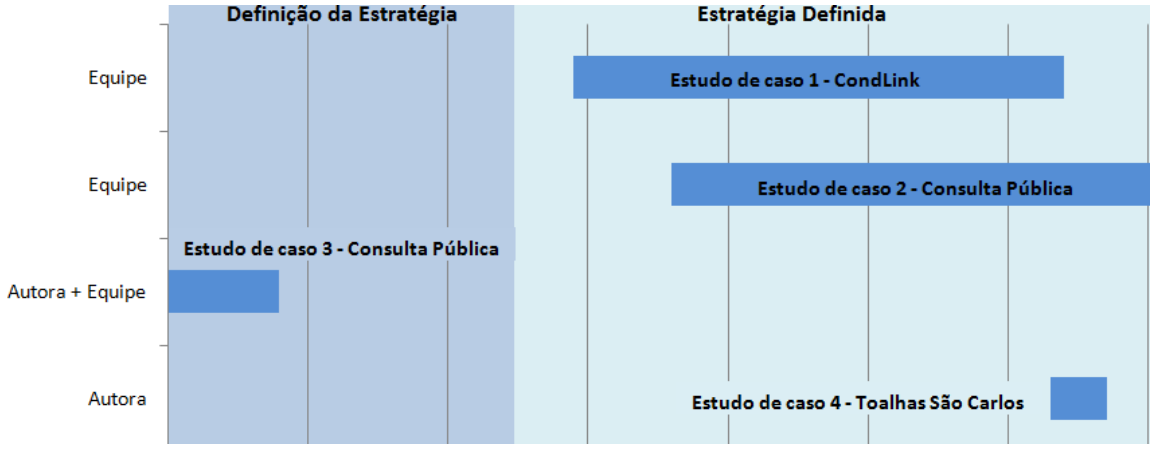


Figura 6.1 – Estudos de caso realizados

O primeiro estudo realizado foi o estudo de caso 3, aplicado no sistema B (Consulta Pública), ainda durante a definição da estratégia, contando com a

supervisão da autora e com o intuito de avaliar a aplicação das atividades de teste em um sistema ainda em desenvolvimento.

O segundo estudo realizado foi o estudo de caso 1, aplicado no sistema A denominado CondLink, somente pela equipe de desenvolvimento, com o objetivo de avaliar a estratégia no planejamento e acompanhamento do trabalho a ser realizado.

O terceiro estudo realizado foi o estudo de caso 3, aplicado no sistema B, pouco tempo depois do início do estudo de caso 1. Também realizado somente pela equipe de desenvolvimento, assim como o estudo de caso 1, foi aplicado com o intuito de avaliar a estratégia no contexto do planejamento de software.

Por fim, o estudo de caso 4, foi aplicado pela autora no sistema C, denominado Toalhas São Carlos, após a definição da estratégia e com o objetivo de avaliar o processo de teste da PW-PlanTe.

6.3 Resultados da aplicação das atividades de planejamento da PW-PlanTe

A seguir, os resultados são apresentados da seguinte forma: o estudo de caso 1 refere-se ao acompanhamento do desenvolvimento do sistema web A. CondLink na empresa Linkway, e o exemplo de aplicação 2 refere-se ao acompanhamento do desenvolvimento do sistema web B. Consulta Pública desenvolvido pela NBS.

6.3.1 Estudo de caso 1 – Empresa Linkway: Sistema CondLink

Esse estudo de caso representa o acompanhamento do desenvolvimento de um sistema web para o gerenciamento e administração de condomínios denominado CondLink. Ele tem por objetivo permitir uma melhor comunicação entre moradores, funcionários, síndicos, conselheiros e administradores de um condomínio, sendo que os problemas levantados por síndicos e administradores de condomínios residenciais e comerciais foram a principal fonte de levantamento de requisitos. O sistema permite, além das funcionalidades convencionais como controle financeiro e

registro de atas, melhorias na comunicação entre os usuários do sistema, a resolução rápida das solicitações realizadas, a reserva dos espaços do condomínio, dentre outras funcionalidades.

O sistema, ainda em desenvolvimento, está sendo construído na linguagem Java, por uma equipe formada por um desenvolvedor, um testador, um gerente e um Product Owner. Foi acompanhado o desenvolvimento de cinco iterações, com duração de uma semana cada. O planejamento das iterações foi realizado por meio da estratégia PW-PlanTe, de forma que durante o desenvolvimento dos IWs houve o constante controle e eventual ajuste do NDE do Time.

Por serem as primeiras iterações em que a equipe aplicou a estratégia PW-PlanTe, o objetivo nesse momento, foi identificar qual era o NDE da Iteração (ou seja, o NDE da equipe), isto é, a relação entre o tempo realmente gasto para desenvolvimento e o tamanho do trabalho realizado. Até o momento da aplicação da estratégia a equipe não possuía um controle rigoroso do tempo gasto em desenvolvimento e, portanto, não possuía uma base histórica bem formada que permitisse que o NDE fosse calculado.

A equipe adotava anteriormente ao uso da estratégia, baseado em projetos anteriores, que o valor de seu NDE era de 1 hora/UT. Entretanto, optou-se por aplicar a estratégia durante algumas iterações de forma a obter valores mais reais. Com esse valor de NDE estimado, se a equipe despende, em média, 8 horas de trabalho por dia (5 horas em desenvolvimento e 3 horas em teste), o que resulta em 40 horas semanais, a capacidade de pontos a serem realizados em uma iteração seria de 40 pontos.

Nesse projeto usou-se, na aplicação da PW-PlanTe os métodos de estimativa ágeis *Planning Poker* em conjunto com a sequência de Fibonacci. A lista de IWs, registrada na FireScrum no *Uncommited Backlog*, foi priorizada antes das reuniões pelo Product Owner, que colocou no topo da pilha os IWs que deveriam ser desenvolvidos primeiro. Essa lista priorizada contava com 14 IWs que foram ordenados, da esquerda (maior prioridade) para a direita (menor prioridade), com os seguintes pontos atribuídos a cada IW: {1, 3, 13, 2, 8, 8, 8, 2, 5, 13, 8, 3, 2, 2}.

Na Tabela 6.2, são apresentadas cinco iterações realizadas para desenvolvimento dos IWs do sistema. Cada linha da tabela corresponde a um dos IWs que estão organizados em iterações.

Ao aplicar a PW-PlanTe, a Tabela 6.2 deve ser construída iterativamente, à medida que os IWs são desenvolvidos. Os valores da tabela correspondem aos seguintes itens: Iteração a ser desenvolvida (PW); Itens de Trabalho (IW – *Item of Work*); Pontos por IW (individual e acumulado); Tempo consumido em desenvolvimento (individual e acumulado); Tempo consumido em teste (individual e acumulado); Tempo total gasto em desenvolvimento e teste; NDE de desenvolvimento obtido por meio da divisão do Tempo acumulado em desenvolvimento pelos Pontos por IW acumulados; NDE de testes obtido por meio da divisão do Tempo total acumulado em teste pelos Pontos por IW acumulados; NDE Médio obtido pelo cálculo do Tempo total gasto dividido pelos Pontos por IW acumulados; Pontos planejados para a iteração que corresponde à soma dos Pontos por IW (individual); Pontos realizados na iteração que corresponde à soma dos pontos referentes aos IWs que foram concluídos naquela iteração; Pontos Recomendados, correspondente à carga horária padrão da iteração, geralmente de 40 horas, dividida pelo NDE da iteração anterior; Pontos realizados acumulados que corresponde ao somatório dos pontos realizados até o momento; NDE da Iteração que corresponde ao Tempo total gasto dividido pelos Pontos realizados acumulados até o momento.

Tabela 6.2 – Acompanhamento da aplicação da Estratégia PW-PlanTe no sistema web CondLink

Iteração (PW)	IW	Pontos por IW		Tempo Desenv.		Tempo Teste		Tempo total	NDE Desen.	NDE Teste	NDE Médio	Pontos Planej.	Pontos Realiz.	Pontos Recom.	Pontos Realiz. Acum.	NDE da Iteração
		Indiv.	Acum.	Indiv.	Acum.	Indiv.	Acum.									
1	1	1	1	1,0	1,0	1,0	1,0	2,0	1,00	1,00	2,00	19	19	19	19	1,79
	2	3	4	6,0	7,0	2,0	3,0	10,0	1,75	0,75	2,50					
	3	13	17	13,0	20,0	3,0	6,0	26,0	1,18	0,35	1,53					
	4	2	19	6,0	26,0	2,0	8,0	34,0	1,37	0,42	1,79					
2	5	8	27	10,0	36,0	2,0	10,0	46,0	1,33	0,37	1,70	18	10	22,34	29	2,14
	6	2	29	15,0	51,0	1,0	11,0	62,0	1,76	0,38	2,14					
	7	8	29	0,0	51,0	0,0	11,0	62,0	1,76	0,38	2,14					
3	8	5	34	5,0	56,0	15,0	26,0	82,0	1,65	0,76	2,41	13	5	9,34	34	2,41
	9	8	34	0,0	56,0	0,0	26,0	82,0	1,65	0,76	2,41					
4	10	13	47	14,0	70,0	15,0	41,0	111,0	1,49	0,87	2,36	24	16	16,6	50	2,44
	11	8	47	0,0	70,0	0,0	41,0	111,0	1,49	0,87	2,36					
	12	3	50	8,0	78,0	3,0	44,0	122,0	1,56	0,88	2,44					
5	13	2	52	8,0	86,0	5,0	59,0	145,0	1,65	0,94	2,60	20	10	16,39	60	2,52
	14	2	52	0,0	86,0	0,0	59,0	145,0	1,65	0,94	2,60					
	15	8	52	0,0	86,0	0,0	59,0	145,0	1,65	0,94	2,60					
	16	8	60	14,5	100,5	1,5	60,5	161,0	1,68	0,84	2,52					

Contrariando a estimativa inicial, de que seria possível realizar 40 pontos em cada iteração, dado que a equipe considerava que seu NDE era de 1 hora/UT, durante a primeira reunião de planejamento, ao se deparar com um processo em

que as etapas estavam bem definidas, a equipe resolveu selecionar 4 IWs para serem implementados na primeira iteração, somando um total de 19 pontos.

Ressalta-se que o simples fato de se tentar sistematizar a atividade de planejamento, e exigir que o tamanho do IW fosse calculado com base em uma técnica de estimativa, isso já mostrou que a equipe fazia um planejamento equivocado, pois ela assumia que seu NDE era igual a 1, mas, por outro lado, não assumia um total de 40 pontos para serem implementados em uma iteração.

O acompanhamento das iterações mostrado na Tabela 6.2 ocorreu da seguinte forma:

- **Iteração 1**:
  - **Alocação dos IWs:**
    - Capacidade da iteração estimada inicialmente: 19 pontos.
    - A lista de IWs no momento era: {1, 3, 13, 2, 8, 8, 8, 2, 5, 8, 13, 3, 2, 2}.
    - Selecionaram-se os IWs de 1 a 4 da lista que correspondiam aos pontos 1, 3, 13 e 2, somando um total de 19 pontos.
  - **Desenvolvimento:**
    - **IW 1:** Ao terminar o primeiro IW o NDE Médio foi 2,0 hora/UT, o que indica que para os 18 pontos restantes para a iteração (Pontos planejados – Pontos por IW acumulados = 19 – 1 = 18) ainda seriam necessárias 36 horas de desenvolvimento (2,0 * 18). Mas, como o total de tempo da iteração era de 40 horas, e até o momento tinham sido consumidas 2 horas (correspondente ao “Tempo total” da Tabela 6.2), ainda existiriam 38 horas disponíveis para a equipe gastar com o desenvolvimento dos IWs que estavam faltando. Assim, o gerente podia assumir que a situação estava sob controle, pois a quantidade trabalho a ser realizada na iteração ainda poderia ser cumprida.
    - **IW 2:** Ao finalizar o segundo IW o NDE Médio foi de 2,5 hora/UT, o que indica que, para os 15 pontos restantes para a iteração (19 – 4 = 15), ainda seriam necessárias 37,5 horas (2,5 * 15). Nesse caso, o gerente e a equipe precisariam ficar atentos, pois

a produtividade da equipe caiu. Considerando esse novo NDE igual a 2,5 para o restante da iteração, a quantidade de trabalho alocada já não poderia ser cumprida, pois já haviam sido gastas 10 horas (Tempo total), restavam somente 30 horas e a equipe precisaria de 37,5 horas, de acordo com a produtividade da iteração, que foi 2,5 hora/UT.

- O **IW 3** e o **IW 4** foram realizados e produtividade da equipe aumentou, o que fez com que, ao finalizar o IW 4, o NDE Médio foi de 1,79 hora/UT e o tempo total gasto foi de 34 horas. Dessa forma, o que aconteceu na realidade foi que, para concluir a quantidade de trabalho elencada para a Iteração 1, foram gastas somente 34 horas das 40 horas disponíveis pela equipe.

- **Planejamento da próxima iteração:**
  - Ao finalizar a Iteração 1 o NDE da Iteração foi 1,79 hora/UT (Tempo total ÷ Pontos Realizados Acumulados = 34 ÷ 19 = 1,79). Com esse valor a **Iteração 2** foi planejada.
  - Com a carga horária de 40 horas determinada pela empresa para cada iteração e o NDE de 1,79, o valor de pontos que poderiam ser realizados na **Iteração 2** seria de, em média, 22,34 pontos (40 horas / 1,79 hora/UT).

– **Iteração 2:**

- **Alocação dos IWs:**
  - Capacidade da iteração: 22,34 pontos.
  - A lista de IWs no momento era: {8, 8, 8, 2, 5, 13, 8, 3, 2, 2}.
  - Para essa iteração os IWs seguintes na lista somariam um total de 24 pontos (8 + 8 + 8). Mas, de acordo com o rendimento da equipe na iteração anterior, NDE = 1,79, a capacidade dessa iteração seria de 22,34 pontos. Assim, a equipe decidiu por selecionar o próximo IW da lista, de 2 pontos, e deixar o de 8 pontos para fazer na próxima iteração.
  - Selecionaram-se os IWs com complexidade de 8, 8 e 2, somando um total de 18 pontos.
- **Desenvolvimento:**

- Essa iteração foi desenvolvida e o NDE da Iteração foi de 2,14 hora/UT.
- Nessa iteração só foram desenvolvidos dois dos três IWs selecionados, somando 10 pontos em um tempo total de 28 horas. Essa situação ocorreu pelo fato do desenvolvedor desse Time ter sido remanejado em alguns momentos para outros projetos da empresa.

- **Planejamento da próxima iteração:**
  - Ao finalizar a Iteração 2 o NDE da Iteração igual a 2,14 hora/UT foi utilizado para estimar a quantidade de pontos que "caberia" na próxima iteração, de acordo com o rendimento até o momento, ou seja, 18,69 pontos (40 horas / 2,14 hora/UT).
  - Entretanto, como o desenvolvedor não estaria presente na empresa durante todo o período da próxima iteração, e sua disponibilidade seria apenas de 20 horas, a quantidade de pontos que poderia ser alocada seria de 9,34 pontos (20 horas / 2,14 hora/UT).

– **Iteração 3:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 9,34 pontos.
    - A lista de IWs no momento era: {8, 5, 8, 8, 13, 3, 2, 2}.
    - Apesar da capacidade da iteração ser de 9,34 pontos, a equipe optou por alocar para a iteração os dois IWs seguintes da lista, somando um total de 13 pontos (8+5).
  - **Desenvolvimento:**
    - Essa iteração foi desenvolvida e o NDE da Iteração foi de 2,41 hora/UT.
    - Nessa iteração só foi desenvolvido o IW de 5 pontos em um tempo total de 20 horas. Como o NDE da equipe subiu para 2,41, um valor ainda maior que o da iteração 2, isso mostra que a produtividade da equipe diminuiu, pois ela demorou mais tempo para realizar cada unidade de trabalho (cada ponto).
  - **Planejamento da próxima iteração:**

- Ao finalizar a Iteração 3 o NDE da Iteração igual a 2,41 hora/UT foi utilizado para estimar a próxima iteração que comportaria então, 16,6 pontos (40 horas / 2,41 hora/UT).

- **Iteração 4:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 16,6 pontos.
    - A lista de IWs no momento era: {8, 8, 8,13, 3, 2, 2}.
    - Nesse momento, durante a reunião, a equipe considerou que sua produtividade estava muito baixa e decidiu elencar para essa iteração 24 pontos, ao invés da capacidade de 16,6 pontos determinada com base no NDE. Assim, foram selecionados os IWs de 8, 13 e 3 pontos,
  - **Desenvolvimento:**
    - Essa iteração foi desenvolvida e o NDE da Iteração foi de 2,44 hora/UT.
    - Nessa iteração, dos três IWs selecionados só foram desenvolvidos dois, somando 16 Pontos realizados num total de 40 horas. Isso mostra que o que havia sido indicado pela estratégia estava correto, ou seja, a equipe só tinha condições de desenvolver os 16,6 pontos, compatíveis com o NDE de 2,41 obtido na iteração 3.
  - **Planejamento da próxima iteração:**
    - Ao finalizar a Iteração 4 o NDE da Iteração igual a 2,44 hora/UT foi utilizado para estimar a iteração 5 que comportaria então 16,39 pontos (40 horas / 2,44 hora/UT).
- **Iteração 5:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 16,39 pontos.
    - A lista de IWs no momento era: {8, 8, 8, 2, 2}.
    - Assim como na iteração anterior, mais uma vez a equipe considerou que estava entregando muito pouco para o cliente e decidiu elencar para a próxima iteração 20 pontos, selecionando os IWs da seguinte forma: 8, 8, 2 e 2.

- **Desenvolvimento:**
  - Essa iteração foi desenvolvida e o NDE da Iteração foi de 2,52 hora/UT.
  - Nessa iteração só foram desenvolvidos dois dos quatro IWs selecionados, somando 10 Pontos realizados, o que indica que a equipe, mais uma vez, não seguiu o que estava sendo indicado pela estratégia e errou novamente.
- **Planejamento da próxima iteração:**
  - Ao finalizar a Iteração 5 o NDE da Iteração foi 2,52 hora/UT. Esse valor seria utilizado para estimar a próxima iteração, que poderia conter 15,87 pontos (40 horas / 2,52 hora/UT).

Ainda restaram 3 IWs (8, 8, 2) da lista de IWs a serem feitos. No entanto, não foi possível coletar os dados do desenvolvimento desses IWs para esse estudo de caso.

A Figura 6.2 resume o que ocorreu durante o uso da estratégia PW-PlanTe nesse sistema. Como a empresa estava usando a estratégia pela primeira vez e isso mudava um pouco sua rotina de trabalho, a equipe não foi capaz de adotar a estratégia sem acomodações decorrentes de diversos fatores, inclusive imprevistos ocorridos em outros sistemas que faziam com que os desenvolvedores tivessem que se deslocar entre os projetos. Dessa forma, a análise que pode ser feita dos resultados ficou prejudicada. O mesmo ocorreu com o segundo estudo, que será apresentado em seguida.

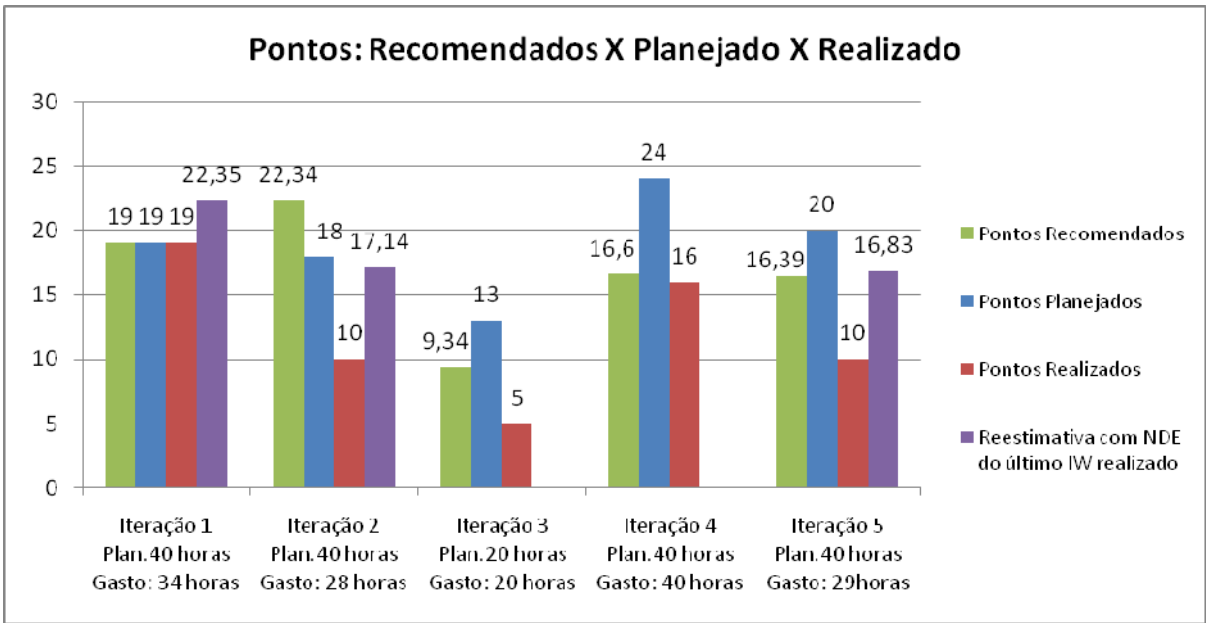


Figura 6.2 – Relação entre os pontos recomendados, planejados e realizados para o sistema CondLink

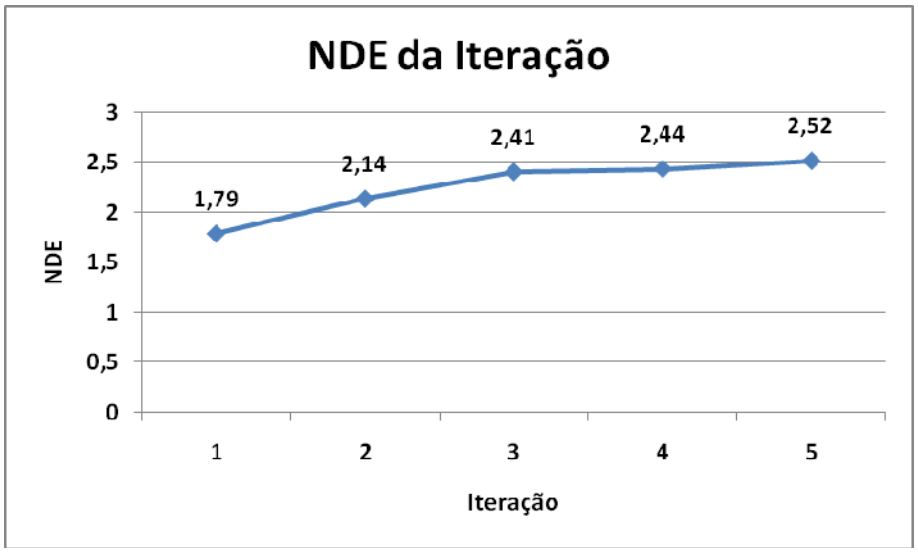


Figura 6.3 – Crescimento do NDE da Iteração

O primeiro fato a ser observado é que das 5 iterações acompanhadas, somente na 3 e 4 foi usado o tempo previsto para a iteração. Nas demais, o tempo usado nas iterações foi sempre inferior ao que estava planejado pela própria empresa.

Assim, analisando primeiramente as iterações 3 e 4 pode-se dizer que na iteração 3, pela estratégia, a previsão foi de que, com o rendimento da equipe na iteração anterior, seria possível realizar mais pontos do que os realizados. A

estimativa era de 9,34 pontos, enquanto foram realizados apenas 5 pontos. Além disso, a própria equipe tinha planejado um número maior ainda (13 pontos) do que os estimados pela estratégia. Observando-se a Figura 6.3, percebe-se que o NDE da iteração 3 foi superior ao NDE da iteração 2. Com base nesses fatos, uma observação que pode ser feita é que a equipe teve mais dificuldade para realizar os pontos da iteração 3 do que os pontos da iteração anterior.

Fazendo a mesma análise para a iteração 4, segundo a estratégia, foram sugeridos 16,6 pontos, a equipe considerou que poderia realizar 24 e, de fato, realizou 16, valor este muito próximo do valor sugerido pela estratégia. Nessa iteração 4 o NDE da equipe foi próximo ao NDE da iteração 3. Com base nesses fatos, percebe-se que, nesse caso, a equipe “teve o rendimento” que se esperava que ela tivesse, de acordo com o que aconteceu na iteração anterior.

Assim, dessas duas iterações em que o tempo planejado para a iteração foi integralmente usado pela equipe, em uma delas, a equipe implementou aproximadamente 53,53% do que foi estimado pela estratégia, enquanto que na outra, ela implementou 96,38% do planejado. Embora tenham sido apenas duas iterações, uma conclusão a que se pode chegar é que está havendo um problema na caracterização da complexidade dos IWs, isto é, na atribuição de pontos que a equipe dá aos IWs, durante a reunião de planejamento de uma nova iteração.

Já na iteração 1, embora a equipe não tenha usado todo o tempo determinado para a iteração, a alocação de IWs foi feita com base na experiência anterior e os pontos dos IWs foram todos realizados. Lembra-se que de acordo com a equipe, a sua proposta inicial era desenvolver quase o dobro de pontos. No entanto, dado o uso da estratégia e o fato disso estar mudando a rotina de trabalho e os desenvolvedores terem que assumir a responsabilidade de fazer uma previsão factível, eles próprios reduziram sua expectativa para quase 50% do que achavam que poderiam realizar. Esse fato mostra, novamente, a inexperiência da equipe em caracterizar a complexidade do trabalho a ser realizado, conforme o que foi, posteriormente, sinalizado nas iterações 3 e 4 descritas anteriormente.

Nas iterações 2 e 5, como o tempo despendido pela equipe para realizar as iterações foi muito inferior ao que estava alocado para elas, não é possível saber o que teria acontecido se a equipe tivesse continuado o trabalho para finalizar os pontos alocados para essas iterações. O que se pode dizer é que, conforme mostra a Figura 6.2, na iteração 2 a equipe teria conseguido realizar quase todos os pontos

planejados por ela para essa iteração. Em relação ao que tinha sido recomendado pela estratégia, a equipe não conseguiria realizar. Isso se deve ao fato de que para o planejamento de uma iteração assume-se o NDE da iteração anterior. Pela Figura 6.3, pode-se perceber que o NDE da iteração 1 foi 83,64% do NDE da iteração 2, o que quer dizer que a equipe se tornou menos produtiva na iteração 2. Esse aspecto deve ser investigado em outros estudos, de forma a se verificar se a maneira de usar a informação do NDE ao longo do desenvolvimento deve ser repensada.

Em resumo, o que se pode dizer, com certeza, é que a implantação de melhoria de processo é realmente uma mudança cultural, por mais que se faça para que o processo seja o menos intrusivo possível.

6.3.2 Estudo de caso 2 – Empresa NBS: Sistema Consulta Pública

Esse estudo representa o acompanhamento do desenvolvimento do sistema web Consulta Pública. Esse sistema, ainda em desenvolvimento, consiste de um portal para a realização de consultas públicas realizadas por prefeituras, com o objetivo de auxiliar na coleta de opiniões dos cidadãos sobre temas de importância. Dessa maneira é possível permitir que a sociedade possa participar da formulação e definição de políticas publicas.

O consulta pública foi construído na linguagem Java, por uma equipe formada por um desenvolvedor, um testador, um gerente e um *Product Owner*.

Assim como no estudo anterior, foi acompanhado o desenvolvimento de cinco iterações, com duração de uma semana cada. Nesse projeto, o planejamento das iterações também foi realizado aplicando a estratégia PW-PlanTe e o desenvolvimento dos IWs do sistema foi acompanhado pelo ajuste do NDE do Time. Para estimativa da complexidade do trabalho a ser desenvolvido, ou seja, dos IWs, foi utilizada a atribuição de pontos por meio do Planning Poker aplicado com a sequência de Fibonacci.

A lista de IWs, registrada na FireScrum no *Uncommited Backlog*, foi priorizada antes das reuniões, pelo *Product Owner*, que colocava no topo da pilha os IWs que deveriam ser desenvolvidos primeiro. Essa lista priorizada era composta de 15 IWs que foram ordenados da esquerda para a direita, dos mais prioritários para os menos prioritários. Inicialmente, a lista tinha a seguinte configuração

{2,3,3,3,3,1,1,3,1,5,8,3,13,3,3}, sendo que os IWs estavam representados na lista pelos pontos (complexidade) que lhes foram atribuídos.

O objetivo da aplicação da estratégia nesse projeto foi, além de fazer uma avaliação da mesma, identificar o NDE de uma Iteração, permitindo saber qual a quantidade de pontos que podem ser elencados para desenvolvimento em uma iteração. A equipe de desenvolvimento também não possuía base de dados histórica e, portanto, os membros da equipe determinaram, com base na experiência, que o NDE incial deveria ser de 1 hora/UT. Além disso, a empresa determinou também que a iteração seria de uma semana, com 20 horas de trabalho por semana. Assim, a quantidade de trabalho alocada para desenvolvimento deveria ser de 4 horas de trabalho por dia (2 horas em desenvolvimento e 2 horas em teste), totalizando as 20 horas semanais. Portanto, com base nessas determinações, a capacidade de pontos a serem realizados em uma iteração seria inicialmente de 20 pontos (20 horas / 1 hora/UT).

Na Tabela 6.3, são apresentadas as cinco iterações que foram acompanhadas durante a realização dos IWs do sistema.

Tabela 6.3 – Acompanhamento da aplicação da Estratégia PW-PlanTe no sistema web Consulta Publica

		Pontos por IW (Fibonacci)		Tempo Desenv.		Tempo Teste		Tempo Total	NDE Desenv.	NDE Testes	NDE Médio	Pontos Plan.	Pontos Realizados	Pontos Recom.	Pontos Realiz. Acum.	NDE da Iteração
Iteração	IW	Indiv.	Acum.	Indiv.	Acum.	Indiv.	Acum.									
1	1	2	2	1,0	1,0	0,0	0,0	1,0	0,50	0,00	0,50	11	11	11,0	11	1,18
	2	3	5	9,0	10,0	0,8	0,8	10,8	2,00	0,16	2,16					
	3	3	8	1,0	11,0	0,2	1,0	12,0	1,38	0,12	1,50					
	4	3	11	0,5	11,5	0,5	1,5	13,0	1,05	0,13	1,18					
2	5	3	14	0,0	11,5	1,0	2,5	14,0	0,82	0,18	1,00	5	4	8,47	15	1,01
	6	1	14	0,0	11,5	0,0	2,5	14,0	0,82	0,18	1,00					
	7	1	15	0,0	11,5	1,2	3,7	15,2	0,77	0,24	1,01					
3	8	3	18	1,0	12,5	0,5	4,2	16,7	0,69	0,23	0,93	17	17	19,80	32	1,21
	9	1	19	0,5	13,0	0,5	4,7	17,7	0,68	0,24	0,93					
	10	5	24	20,0	33,0	0,5	5,2	38,2	1,38	0,21	1,59					
	11	8	32	0,0	33,0	0,5	5,7	38,7	1,03	0,18	1,21					
4	12	3	35	0,0	33,0	0,4	6,1	39,1	0,94	0,17	1,12	22	13	16,56	45	1,23
	13	13	42	7,0	40,0	0,5	6,5	46,5	0,95	0,16	1,11					
	14	3	42	0,0	40,0	0,0	6,5	46,5	0,95	0,16	1,11					
	15	3	45	9,0	49,0	0,0	6,5	55,5	1,09	0,14	1,23					
5	16	3	48	1,6	50,6	0,5	7,0	57,6	1,05	0,15	1,20	7	7	16,21	52	1,21
	17	3	51	3,6	54,2	0,5	7,5	61,7	1,06	0,15	1,21					
	18	1	52	0,0	54,2	1,0	8,5	62,7	1,04	0,16	1,21					

Nesse estudo, por ser também a primeira vez em que a PW-PlanTe estava sendo aplicada, a equipe optou por não alocar para a primeira iteração os 20 pontos obtidos com o NDE = 1 hora/UT, pois ele correspondia a um valor estimado com

base na experiência anterior. Assim, durante a reunião de planejamento na primeira iteração foram selecionados 4 IWs a serem realizados que juntos somavam 11 pontos.

Durante a aplicação da PW-PlanTe o acompanhamento das iterações está retratado na Tabela 6.3 e pode ser interpretado da seguinte forma:

- **Iteração 1**:
  - **Alocação dos IWs:**
    - Capacidade da iteração estimada inicialmente: 11 pontos.
    - A lista de IWs no momento era: {2,3,3,3,3,1,1,3,1,5,8,3,13,3,3}.
    - Selecionaram-se os IWs de 1 a 4 da lista que correspondiam aos pontos 2, 3, 3 e 3, somando um total de 11 pontos.
  - **Desenvolvimento:**
    - **IW 1:** Após a conclusão do primeiro IW o NDE Médio do desenvolvimento foi de 0,5 hora/UT o que indica que para os 9 pontos restantes na iteração (Pontos Planejados – Pontos por IW Acumulados = 11 – 2 = 9) seriam necessárias somente 4,5 horas de desenvolvimento (0,5 * 9) para concluir a iteração.
    - Como esse valor de NDE de 0,5 hora/UT foi muito baixo, o gerente estava alerta para perceber se ele ficaria em torno desse valor, pois em caso afirmativo, seria possível acrescentar mais IWs à iteração.
    - **IW 2:** Ao finalizar o segundo IW, o mesmo cálculo foi efetuado, e o valor de NDE 2,16 hora/UT foi encontrado, contrariando a expectativa criada com a conclusão do IW anterior. Considerando esse novo valor de NDE e considerando que ainda havia 6 pontos para serem implementados nessa iteração, isso implicava em mais 12,96 horas (2,16 * 6 pontos) de trabalho pela frente.
    - Observa-se que como o total de tempo elencado para a iteração era de 20 horas, e até a conclusão do IW2 já tinham sido consumidas 10,8 horas, o gerente já sabia que se a produtividade continuasse essa, os IWs alocados para essa iteração não seriam realizados. Ainda eram necessárias 12,96

horas de trabalho e existiam apenas 9,2 horas disponíveis. Nesse momento, o gerente precisou alertar a equipe sobre o possível atraso.

- **IW3:** Ao finalizar o IW 3, o NDE Médio foi de 1,50 hora/UT. Considerando esse valor e os pontos que ainda faltavam ser implementados (3 pontos), a equipe precisaria ainda de 4,5 horas para concluir o desenvolvimento. Como até então haviam sido consumidas 12 horas de trabalho foi possível perceber que os IWs elencados para essa iteração poderiam ser entregues dentro do planejado.
- **IW 4:** Ao finalizar o IW 4 o NDE Médio foi de 1,18 hora/UT e foram consumidas em toda iteração 13 horas das 20 alocadas para a iteração.

- **Planejamento da próxima iteração:**
  - Ao finalizar a Iteração 1 o NDE da Iteração foi 1,18 hora/UT (Tempo total ÷ Pontos Realizados Acumulados = 13 horas / 11 pontos = 1,18). Esse valor foi utilizado para planejar o PW da próxima iteração.
  - Com a carga horária de 20 horas para cada iteração e o NDE de 1,18, o valor de pontos que poderiam ser realizados na próxima iteração seria de 16,94 pontos (20 horas / 1,18 hora/UT).
  - Como na próxima iteração sabia-se que o desenvolvedor não poderia estar presente durante as 20 horas da iteração, foi determinado que a iteração teria 10 horas. Assim, a quantidade de pontos que caberia nessa iteração seria de 8,47 pontos (10 horas / 1,18 hora/UT).

- **Iteração 2:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 8,47 pontos.
    - A lista de IWs no momento era: {3,1,1,3,1,5,8,3,13,3,3}.
    - Os IWs selecionados para a iteração foram {3,1,1}
  - **Desenvolvimento:**
    - Para essa iteração foram selecionados somente 3 IWs somando um total de 5 pontos para serem realizados.

- Ao final do desenvolvimento o NDE encontrado foi 1,01 hora/UT (15,2 / 15) sendo que apenas 4 dos 5 pontos foram realizados em um total de 2,2 horas.

- **Planejamento da próxima iteração:**
  - Com o NDE de 1,08 hora/UT e considerando o tempo de 20 horas da iteração, a sugestão de pontos a serem realizados na iteração 3 foi de 19,8 pontos (20 horas / 1,01 hora/UT).
  - Com base na lista de IWs e na estimativa de pontos que caberiam na iteração, de acordo com a produtividade refletida pelo NDE, definiu-se que a próxima iteração teria 17 pontos

- **Iteração 3:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 19,8 pontos.
    - A lista de IWs no momento era: {1,3,1,5,8,3,13,3,3}.
    - Os IWs selecionados para a iteração foram {3, 1, 5, 8 } somando um total de 17 pontos.
  - **Desenvolvimento:**
    - Os 17 pontos elencados para a Iteração foram desenvolvidos e foram gastas 23,5 horas.
    - O valor de NDE encontrado nessa iteração foi de 1,21 hora/UT (38,7 / 32).
  - **Planejamento da próxima iteração:**
    - Com o NDE de 1,21 a próxima iteração poderia comportar 16,56 pontos (20 / 1,21).
    - Entretanto, como a equipe não estava satisfeita com seu rendimento e com a quantidade de produto entregue ao cliente, elencar uma quantidade de 22 pontos para serem realizados na próxima iteração.
- **Iteração 4:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 16,56 pontos.
    - A lista de IWs no momento era: {1,3,13,3,3}.

- Os IWs elencados para a iteração foram {3, 13, 3 e 3} somando 22 pontos. O IW de complexidade 1, mesmo sendo o próximo na lista de prioridade foi deixado para a próxima iteração.

- **Desenvolvimento:**
  - Essa iteração foi desenvolvida e o NDE da Iteração foi de 1,23 hora/UT (55,5 / 45).
  - Nessa iteração só foram desenvolvidos dois dos três IWs selecionados, somando 10 Pontos. Entretanto, o IW de complexidade igual a 13 pontos não pode ser concluído e a equipe considerou que mais de 50% desse item foi cumprido, ou seja, em torno de 7 pontos que, somando com o IW de 3 pontos resultou em um total de 10 Pontos Realizados na iteração, em um tempo total de 16,7 horas.
  - Esse fato indica que a equipe errou em ter estimado o valor de 22 pontos para ser realizado na iteração.
- **Planejamento da próxima iteração:**
  - Com o NDE da Iteração igual a 1,23 hora/UT a próxima iteração poderia comportar 16,21 pontos (20 horas / 1,23 hora/UT).

- **Iteração 5:**
  - **Alocação dos IWs:**
    - Capacidade da iteração: 16,21 pontos.
    - A lista de IWs no momento era: {1, 3, 3, 6}, sendo que o IW de peso 6 corresponde à parte do IW de peso 13 iniciado na iteração anterior e não concluído.
    - Foram selecionados os IWs {1, 3 e 3} para esta iteração e o desenvolvimento do IW de 6 pontos não pode ser acompanhado neste estudo de caso.
  - **Desenvolvimento:**
    - Completado o desenvolvimento o NDE da Iteração foi de 1,21 hora/UT, os 7 pontos elencados para a iteração foram desenvolvidos em um total de 7,2 horas.
  - **Planejamento da próxima iteração:**

- Todos os IWs elencados para esta iteração foram desenvolvidos e o total de pontos que poderia ser elencado para a próxima iteração seria de 16,52 pontos (20 horas / 1,21 hora/UT).

A Figura 6.4 resume o que ocorreu durante o uso da estratégia PW-PlanTe nesse sistema. Assim como no estudo de caso anterior, a empresa NBS estava usando a estratégia pela primeira vez e isso interferia na sua rotina de trabalho. Dessa forma, a análise também ficou prejudicada uma vez que imprevistos como deslocamentos de desenvolvedores entre os projetos também ocorreu e a análise que pode ser feita dos resultados ficou prejudicada.

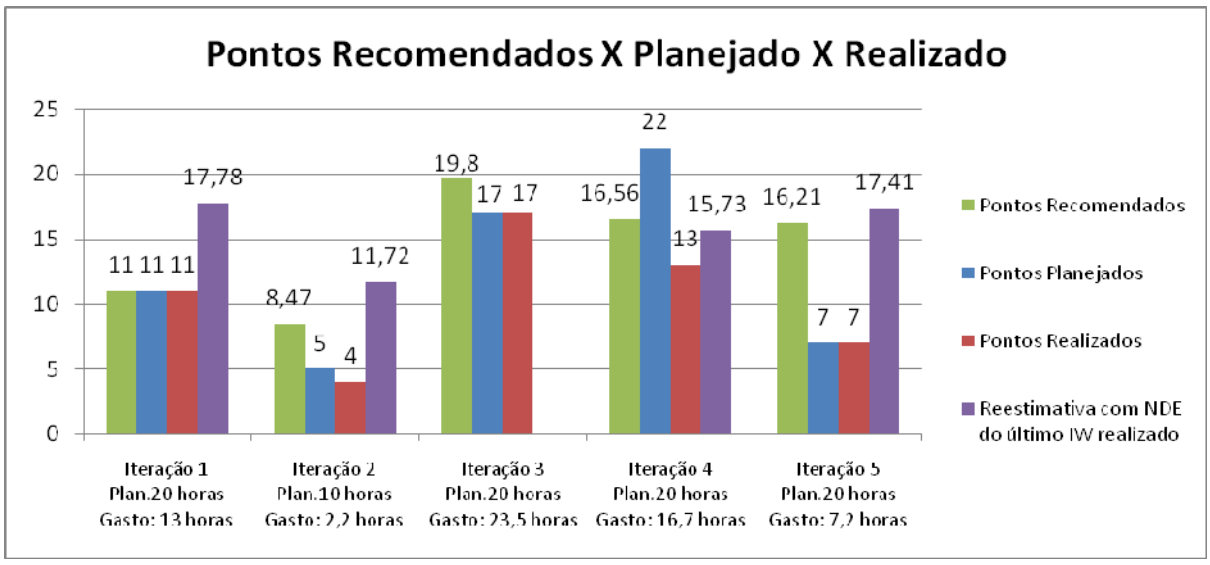


Figura 6.4 – Relação entre os pontos recomendados, planejados e realizados para o sistema Consulta Pública

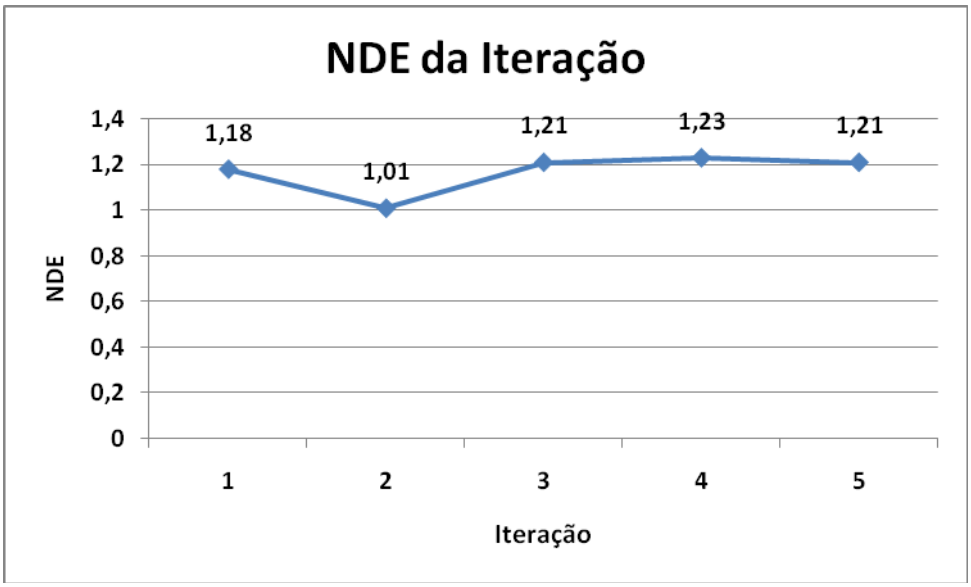


Figura 6.5 – Acompanhamento do NDE da Iteração

O primeiro fato a ser observado é que em nenhuma das 5 iterações acompanhadas, foi usado o tempo previsto para a iteração planejado pela empresa, exceto na iteração 3 em que o tempo inclusive extrapolou o previsto.

Nas iterações 1, 3 e 5, a equipe conseguiu realizar a quantidade de pontos selecionados para a iteração, entretanto, nas iterações 1 e 5 houve sobra de tempo. Como o tempo despendido pela equipe para realizar essas duas iterações foi muito inferior ao que estava alocado para elas, não é possível saber o que teria acontecido se a equipe tivesse continuado o trabalho nessas horas restantes. O que é possível dizer é que, caso o NDE da equipe, calculado ao finalizar o desenvolvimento dos pontos nas iterações 1 e 5, fosse utilizado para calcular quantos pontos a mais seria possível desenvolver com as horas que restavam, essa quantidade de pontos seria maior inclusive do que a quantidade recomendada pela estratégia, a qual estava acima da quantidade que a equipe considerava viável realizar.

Na iteração 3, a equipe conseguiu realizar a quantidade de pontos alocada, entretanto extrapolou a quantidade de horas alocadas para a iteração. O valor de pontos realizado foi inclusive menor que o recomendado pela estratégia. Isso se dá pelo fato de que a capacidade em pontos a ser realizada na iteração é calculada com base no NDE da iteração anterior. Assim, como o NDE da equipe aumentou (Figura 6.5) da iteração 2 para a iteração 3, é justificável que a equipe tenha conseguido fazer menos pontos que o recomendado.

Com a análise dessas três iterações, é possível observar que a equipe ainda não possui uma maturidade na atribuição da complexidade dos pontos a serem realizados em uma iteração. Em alguns momentos a quantidade de pontos selecionada para a iteração é menor do que a equipe consegue fazer (iterações 1 e 5) e em outros momentos maior (iteração 3).

Analisando as iterações 2 e 4 a quantidade de horas consumidas na iteração também foi menor do que o planejado pela empresa, e tanto a capacidade de pontos planejados como recomendados pela estratégia não foram cumpridos. Entretanto, se com a quantidade de horas que sobraram nas iterações fosse calculado a quantidade de pontos que ainda caberia na iteração, com base no NDE do último IW realizado, é possível notar na Figura 6.4 que caberiam mais pontos nessas iterações. Na iteração 2, com o NDE do último IW realizado, caberiam inclusive mais pontos do que o recomendado pela estratégia. Justifica-se tal fato pelo NDE da iteração 2, usado para realizar este cálculo com o tempo restante, ter sido menor que o da iteração 1, usado como base para o planejamento inicial da iteração, como pode ser visualizado na Figura 6.5. Na iteração 4, calculando quantos pontos caberiam ainda na iteração com as horas restantes o valor encontrado é maior do que o que foi realizado e menor do que o recomendado. Isso ocorre também devido ao fato do NDE dessa iteração ser um pouco maior que o da iteração 3 anterior, ou seja, a equipe teve mais dificuldade para realizar os pontos da iteração 4 do que os pontos da iteração 3.

Na iteração 4 aproximadamente 58,55% do que foi estimado pela estratégia foi realizado. Assim como no estudo anterior uma conclusão a que se pode chegar é que está havendo um problema na caracterização da complexidade dos IWs, isto é, na atribuição de pontos que a equipe dá aos IWs, durante a reunião de planejamento de uma nova iteração.

Com estes resultados, assim como discutido no estudo apresentado na subseção anterior, é possível afirmar que a mudança cultural é uma questão que torna a melhoria de processo difícil de ser realizada. Além disso, é possível afirmar também, que a qualidade dos resultados da aplicação da estratégia está bastante ligada à maturidade da equipe em estimar a complexidade do trabalho a ser desenvolvido.

6.4 Resultados da aplicação das atividades de teste da PW-PlanTe

Os estudos 3 e 4 referem-se à aplicação da estratégia no que diz respeito ao processo de teste criado, com o intuito de mostrar a sua efetividade em comparação aos testes realizados informalmente, antes da implantação da estratégia ou, em alguns casos, comparados à completa ausência de teste em alguns projetos. O primeiro estudo refere-se à aplicação do processo no sistema *web* Consulta Pública, desenvolvido pela NBS. O objetivo desse estudo foi realizar um comparativo entre os dados de cobertura e defeitos coletados com a aplicação do processo de teste da estratégia PW-PlanTe e o processo informal realizado na empresa. O segundo estudo refere-se à aplicação do processo no sistema *web* Toalhas São Carlos, desenvolvido pela Linkway. Esse estudo teve por objetivo mostrar a aplicação do processo de teste em um sistema já desenvolvido e já no ambiente de produção, mas que apresentou defeitos que só foram descobertos durante a execução do sistema pelo cliente.

6.4.1 Estudo de caso 3 – Comparação entre o processo de teste formalizado e informal

Este estudo foi realizado durante o desenvolvimento do sistema Consulta Pública. Para realizar esse estudo foram preparadas duas versões do sistema a ser testado, instrumentadas por meio da ferramenta Cobertura. Em uma versão foram aplicadas as atividades de teste propostas pelo processo de testes da estratégia PW-PlanTe e na outra versão foram realizados testes informalmente, da maneira como vinham sendo realizados na empresa, de forma *ad-hoc*. Essa instrumentação do sistema permitiu avaliar a cobertura alcançada com o processo informal de teste e com o processo de teste proposto na PW-PlanTe. Os resultados obtidos dessas atividades foram comparados e divergem quanto à quantidade de linhas cobertas, defeitos encontrados e número de casos de teste aplicados.

O processo informal de teste consistia do informe, por e-mail, feito pelo desenvolvedor à pessoa responsável pelo teste, de que o sistema estava pronto

para ser testado. Nesse e-mail eram descritas as funcionalidades que deveriam ser testadas e também algumas informações para realização dos testes, tais como: logins, senhas, endereço para acesso ao sistema, etc.

Dessa forma, o responsável por executar o teste, que na maioria das vezes não havia tido contato com o sistema ainda e não havia acompanhado o desenvolvimento junto à equipe, seguia as instruções contidas no e-mail. Ele executava os passos necessários para testar as funcionalidades disponibilizadas e reportava, também por e-mail, os problemas encontrados.

A atividade de teste conforme planejado (informal e pela PW-PlanTe) foi aplicada em duas iterações do desenvolvimento do sistema Consulta Pública. As funcionalidades referem-se ao módulo de envio de mala direta que pode ser feito pelos perfis dos usuários *Root* e Administrador, os quais enviam a mala direta aos usuários Contribuidores das consultas públicas já realizadas no sistema.

O processo de teste da PW-PlanTe refere-se ao processo automatizado de teste descrito no Capítulo 5, realizado com o auxilio da ferramenta de automatização Selenium e das ferramentas FireScrum e Cobertura. Nesse processo, primeiramente foi realizado o registro dos testes de aceitação para as funcionalidades a serem testadas (Etapa A). Foram estabelecidos 10 cenários de teste para a funcionalidade relacionada ao usuário Administrador e 6 para a funcionalidade relacionada ao usuário *Root*.

Para cada um dos cenários de aceitação foram criados *scripts* de teste usando a Selenium (Etapa B) e, nesse momento, antes da execução dos *scripts* de teste é que a instrumentação utilizando a ferramenta Cobertura foi realizada. Após a instrumentação, os scripts de teste criados foram executados (Etapa C) e os defeitos foram registrados (Etapa D). Uma vez executados os *scripts* os relatórios de cobertura podem ser gerados e analisados (Etapas E e F).

Os resultados da aplicação do processo informal e do processo de teste da PW-PlanTe podem ser visualizados na Tabela 6.4.

Tabela 6.4 – Resultados da aplicação dos testes segundo a PW-PlanTe comparado ao processo informal

Processo	Funcionalidade	Quantidade de casos de teste	Linhas cobertas nos métodos (Qtd. de linhas cobertas e qtd. de linhas existentes)	Quantidade de defeitos	Tempo (horas)
Informal	Administrador: Enviar mala direta para contribuintes de uma consulta	1	22 de 43	2	1:00
	Administrador: Enviar mala direta para todos os contribuintes de um órgão	1	29 de 47	0	
	Root: Enviar mala direta para todos os contribuintes de um órgão	1	33 de 50	1	
	Root: Enviar mala direta para todos os contribuintes de todos os órgãos	1	26 de 40	0	
Automatizado (PW-PlanTe)	Administrador: Enviar mala direta para contribuintes de uma consulta	5	38 de 43	4	15:44
	Administrador: Enviar mala direta para todos os contribuintes de um órgão	5	47 de 47	0	
	Root: Enviar mala direta para todos os contribuintes de um órgão	6	45 de 50	4	
	Root: Enviar mala direta para todos os contribuintes de todos os órgãos	1	32 de 40	0	

Como pode ser observado na Tabela 6.4 há uma grande variação entre o processo informal e o processo de testes da PW-PlanTe com relação à quantidade de testes realizados (Figura 6.6), à quantidade de defeitos (Figura 6.7) encontrados, e também à quantidade de linhas cobertas (Figura 6.8).

Enquanto que com o processo informal o testador aplicou somente um caso de teste para cada funcionalidade, no processo formalizado a quantidade de casos de teste foi bem maior exceto para a funcionalidade “Root: Enviar mala direta para todos os contribuintes de todos os órgãos” como mostra a Figura 6.6.

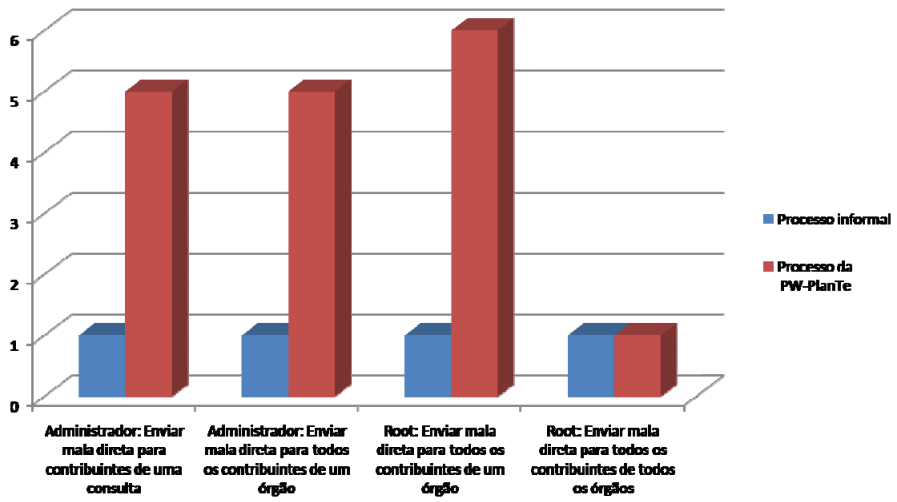


Figura 6.6 – Comparação da quantidade de casos de teste elaborados para o módulo Mala Direta do sistema Consulta Pública

Na Figura 6.7 é possível observar que a quantidade de defeitos identificados pelo processo da PW-PlanTe foi maior do que a encontrada pelo processo informal. Os defeitos identificados com o processo formalizado referem-se a situações que não foram exploradas pelo testador, no processo informal. Ressalta-se que como o processo da PW-PlanTe requer que o testador cadastre os cenários de teste na FireScrum, esse fato força o testador a pensar em diversas situações que no processo informal não são exploradas pois neste último caso o testador se baseia apenas no e-mail recebido pelo desenvolvedor.

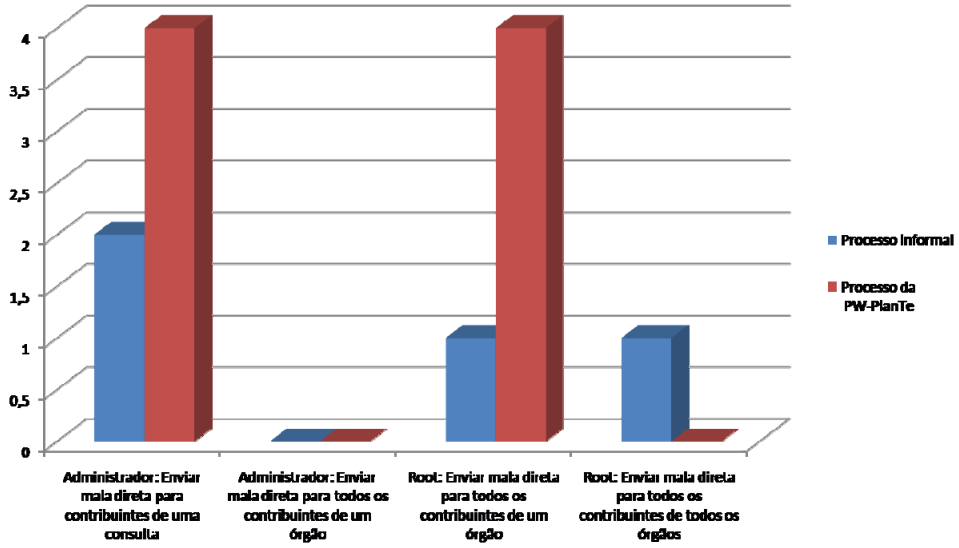


Figura 6.7 – Comparação da quantidade de defeitos encontrados no módulo Mala Direta do sistema Consulta Pública

Na Figura 6.8 é possível observar que em todas as situações houve diferença entre a quantidade de linhas cobertas com a aplicação do processo informal e do processo da PW-PlanTe nos métodos referentes às funcionalidades avaliadas.

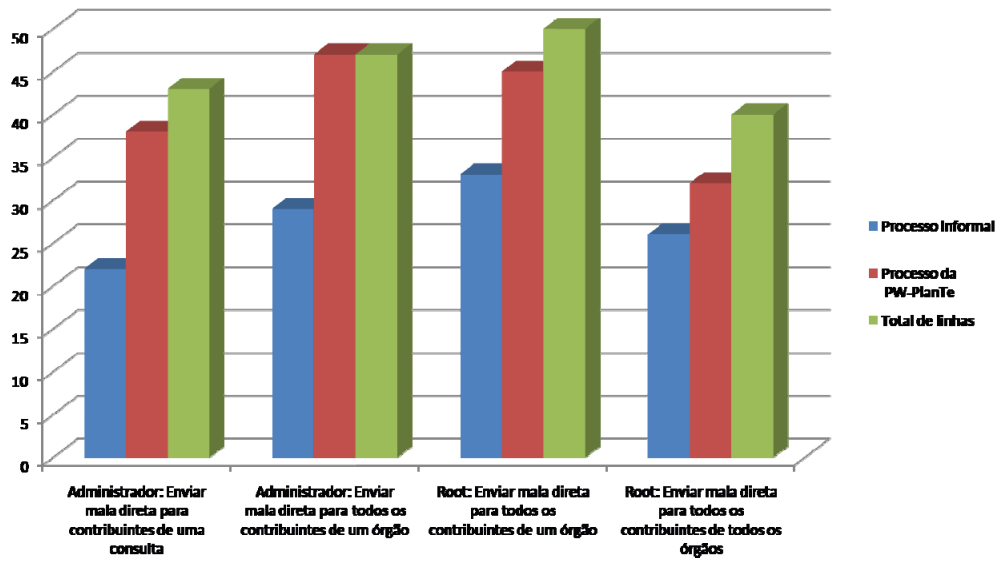


Figura 6.8 – Comparação da cobertura dos testes realizados no módulo Mala Direta do sistema Consulta Pública

O processo informal aplicado se caracterizou por ser realizado sem muito planejamento, apenas com o responsável pelos testes acessando o sistema e verificando se a funcionalidade básica esperada estava funcionando. Dessa forma, o número de cenários executados para cada uma das funcionalidades foi restrito gerando, por vezes, uma cobertura e quantidade de defeitos menor em relação ao processo formalizado. Considerando todas as funcionalidades testadas a diferença as coberturas foram de 90% no processo da PW-PlanTe e 61% no processo informal.

Como exemplo, considerando o cenário “Root: Enviar mala direta para todos os contribuintes de um órgão” da Tabela 6.5 pode-se observar o seguinte: enquanto no processo informal a quantidade de casos de teste e de defeitos foi igual a 1, e a cobertura foi de 33 das 50 linhas do método, no Processo Automatizado de Teste foram criados 6 cenários de teste, que resultaram na descoberta de 4 defeitos e na cobertura de 45 entre 50 linhas desenvolvidas.

Considerando o mesmo cenário relatado no exemplo anterior, o testador não registrou os casos de teste, mas somente anotou o problema ocorrido, informando o fato para os desenvolvedores. Segundo o testador o problema identificado na execução dessa funcionalidade foi relatado da seguinte forma: “Ao enviar mala direta para todos os contribuintes como *root* a resposta é muito lenta e não aparece mensagem dizendo se o e-mail foi enviado ou não”.

Ao finalizar a execução dos testes no processo informal, foram gerados relatórios de cobertura da execução dos testes para comparar com a execução do processo formalizado.

No processo de teste formalizado os cenários de teste criados foram registrados na FireScum, conforme pode ser visualizado na Tabela 6.6, na qual são exibidos os seis cenários criados para a funcionalidade “Root: Enviar mala direta para todos os contribuintes de um órgão”.

Tabela 6.5 – Cenários de teste para a funcionalidade “Root: Enviar mala direta para todos os contribuintes de um órgão”

Passo	Ação/Procedimento	Resultado Esperado
1	Ao selecionar opção enviar e-mail para todos os contribuintes a interface de e-mail deverá ser aberta e os e-mail deverão ser enviados com sucesso	Sucesso
2	Tendo selecionado um órgão mostrado, ao selecionar a opção mala direta será mostrada interface para envio do e-mail para os contribuintes daquele órgão	Sucesso
3	Ao selecionar opção enviar e-mail para todos os contribuintes a interface de e-mail deverá ser aberta e os e-mails não serão enviados com sucesso	Falha (para realizar esse teste, é necessário falhar o servidor smtp antes do primeiro e-mail ser enviado)
4	Tendo selecionado uma órgão mostrado, ao selecionar a opção mala direta será mostrada interface para envio do e-mail para os contribuintes daquele órgão e os e-mails não serão enviados com sucesso	Falha (para realizar esse teste, é necessário falhar o servidor smtp antes do primeiro e-mail ser enviado)
5	Ao selecionar opção enviar e-mail para todos os contribuintes a interface de e-mail deverá ser aberta e alguns e-mails não serão enviados com sucesso	Falha (para realizar esse teste, é necessário falhar o servidor smtp na metade do envio dos e-mails)
6	Tendo selecionado um órgão mostrado, ao selecionar a opção mala direta será mostrada interface para envio do e-mail para os contribuintes daquele órgão e alguns e-mails não serão enviados com sucesso	Falha (para realizar esse teste, é necessário falhar o servidor smtp na metade do envio dos e-mails)

Para cada um desses cenários foram criados scripts de teste na ferramenta Selenium. O script elaborado para o cenário 1 pode ser visualizado na Figura 6.9. Nesse *script*, um usuário do sistema com perfil de *root* foi utilizado para *logar* no sistema (Figura 6.9 – A) e selecionar a opção “Enviar e-mail para todos os contribuintes de uma consulta” (Figura 6.9 – B). O resultado esperado era que a interface de e-mail fosse aberta e os e-mails fossem enviados com sucesso exibindo na tela a mensagem “E-mails enviados com sucesso!” (Figura 6.9 – C).

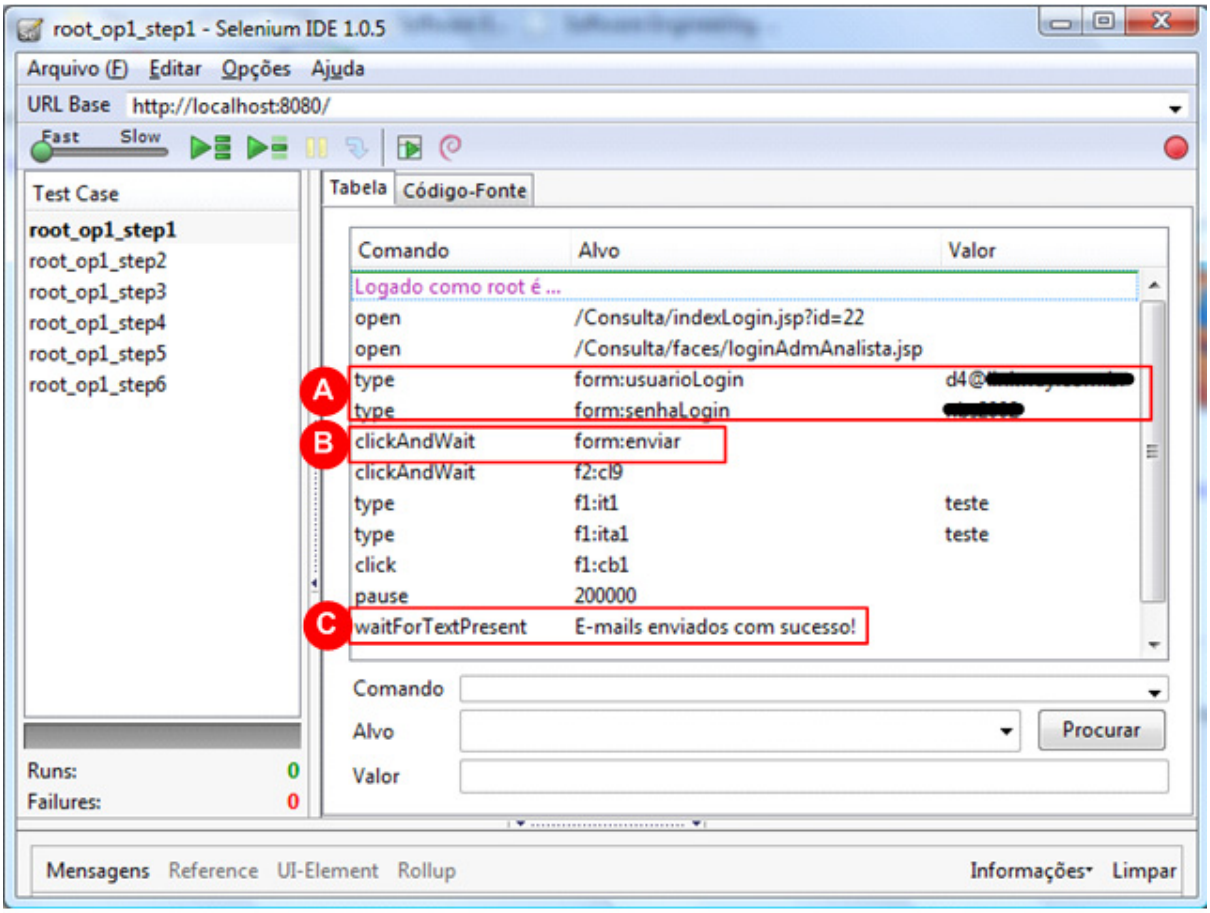


Figura 6.9 – *Script* de teste registrado na Selenium IDE

Para executar esse teste o *script* foi gravado como uma sequência de passos realizados na tela do sistema e repetido durante a Etapa C de Execução de Scripts de Teste. Os defeitos encontrados por meio desse processo foram registrados pelo testador, na FireScrum, no módulo *BugTracking,* para correção posterior (Etapa D – Registro de Defeitos). Tais defeitos referiam-se ao não envio do e-mail, à não exibição de confirmação de envio do e-mail e à retenção dos e-mails no servidor o que impedia que os destinatários recebessem os e-mails do endereço de e-mail utilizado como remetente.

Após a realização desses testes, foi analisada a cobertura (Etapa E) observando se os testes executados realmente exercitaram as instruções relacionadas à funcionalidade testada. Na Figura 6.10 observa-se que a linha 162 (na primeira coluna da figura), que contém a instrução *addSucessMessage*(“E-mails enviados com sucesso!”) está marcada de verde, o que indica que essa instrução foi exercitada. As próximas instruções, também marcadas como executadas pela

ferramenta de cobertura, exibidas nas linhas 164 e 166, representam instruções que seriam cobertas respectivamente por meio dos cenários de teste 4 e 5 da Tabela 6.5. Tais cenários representam as situações nas quais as mensagens seriam “Erro! Os e-mails não puderam ser enviados!” e “Erro! Alguns e-mails não puderam ser enviados!”. Para exercitar esses cenários foi necessário simular uma falha no servidor de envio de e-mails SMTP. Por ser uma situação atípica e pelo fato do testador não visualizar no código que a instrução não foi exercitada, esse cenário acaba não sendo cogitado num teste informal.

```
122   1     // envia emails para todos os contribuintes cadastrados
123   1     public String paraContribuintesDoOrgao() {
124  11         if (idOrgao != null) {  // se for true, eh porque o metodo foi acessado da pagina maladireta.jsp
125  12             httpSession.setAttribute("idOrgao", idOrgao);
126  21         }
127  11         resetAtributos();
128  11         ConsultaController consultaController = new ConsultaController();
129  12         List<Consulta> listaDeConsultasAtivasEEncerradas = new ArrayList<Consulta>();
130  11         List<Contribuicao> listaDeContribuicoes = new ArrayList<Contribuicao>();
131              try {
132  12             listaDeConsultasAtivasEEncerradas.addAll(consultaController.getConsultasAtivasPorOrgao());
133  11             listaDeConsultasAtivasEEncerradas.addAll(consultaController.getConsultasFinalizadasPorOrgao());
134   1
135  12             for (Consulta consultas : listaDeConsultasAtivasEEncerradas) {
136 174                 listaDeContribuicoes.addAll(consultas.getContribuicoes());
137   1             }
138   1
139  11             List<String> emails = concatenaEmails(listaDeContribuicoes);
140   1
141   1             // loop para enviar os itens (strings) da lista emails
142  12             for (String item : emails) {
143   2                 try { // primeira tentativa de enviar os emails do item
144  51                     enviarEmail(item, getTitulo(), getTexto());
145   7                 } catch (Exception ex) {
146   1                     try { // segunda tentativa de enviar os emails do item
147   7                         enviarEmail(item, getTitulo(), getTexto());
148   8                     } catch (Exception e) {
149   1                         // se nao for possivel enviar na segunda tentativa, guarda os emails que nao foram enviados
150   7                         listaNaoEnviados.add(item);
151   8                         setErroAoEnviarEmails(true);
152   7                         setEmailsNaoEnviados("");
153  16                         for (int i = 0; i < listaNaoEnviados.size(); i++) {
154   9                             setEmailsNaoEnviados(getEmailsNaoEnviados() + listaNaoEnviados.get(i));
155                          }
156   7                         e.printStackTrace();
157   0                     }
158   7                     ex.printStackTrace();
159  93                 }
160   0             }
161  12             if (listaNaoEnviados.size() == 0) {
162   6                 addSuccessMessage("E-mails enviados com sucesso!");
163   6             } else if (listaNaoEnviados.size() == emails.size()) {
164   4                 addMessage("Erro! Os e-mails não puderam ser enviados!");
165              } else {
166   2                 addMessage("Erro! Alguns e-mails não puderam ser enviados!");
167   0             }
168   0         } catch (Exception ex) {
169   0             ex.printStackTrace();
170   0             return null;
171  11         }
172  11         return null;
173          }
```

Figura 6.10 – Relatório de cobertura dos testes realizados com o processo formalizado

Na Figura 6.11, que corresponde à execução informal dos testes é possível observar que a instrução que exibe a mensagem “E-mails enviados com sucesso!” foi executada (linha 152). As outras instruções relativas às demais possibilidades

associadas ao envio do e-mail não foram exercitadas, como pode ser visto nas linhas 154 e 156 marcadas em vermelho. Entretanto, como só houve uma tentativa de exercitar essa funcionalidade poucas instruções foram exercitadas no método. Outro ponto a se observar é que, situações que foram pensadas no processo formalizado, como ser necessária a parada do servidor de e-mails, para simular o não-envio de e-mails, no processo informal não foram nem sequer cogitadas.

```
112         // envia emails para todos os contribuintes cadastrados
113         public String paraContribuintesDoOrgao() {
114  1          if (idOrgao != null) {  // se for true, eh porque o metodo foi acessado da pagina maladireta.jsp
115  1              httpSession.setAttribute("idOrgao", idOrgao);
116             }
117  1          resetAtributos();
118  1          ConsultaController consultaController = new ConsultaController();
119  1          List<Consulta> listaDeConsultasAtivasEEncerradas = new ArrayList<Consulta>();
120  1          List<Contribuicao> listaDeContribuicoes = new ArrayList<Contribuicao>();
121             try {
122  1              listaDeConsultasAtivasEEncerradas.addAll(consultaController.getConsultasAtivasPorOrgao());
123  1              listaDeConsultasAtivasEEncerradas.addAll(consultaController.getConsultasFinalizadasPorOrgao());
124
125  1              for (Consulta consultas : listaDeConsultasAtivasEEncerradas) {
126 19                 listaDeContribuicoes.addAll(consultas.getContribuicoes());
127               }
128
129  1              List<String> emails = concatenaEmails(listaDeContribuicoes);
130
131              // loop para enviar os itens (strings) da lista emails
132  1              for (String item : emails) {
133                   try { // primeira tentativa de enviar os emails do item
134  1                   enviarEmail(item, getTitulo(), getTexto());
135  0                } catch (Exception ex) {
136                       try { // segunda tentativa de enviar os emails do item
137  0                        enviarEmail(item, getTitulo(), getTexto());
138  0                    } catch (Exception e) {
139                           // se nao for possivel enviar na segunda tentativa, guarda os emails que nao foram enviados
140  0                        listaNaoEnviados.add(item);
141  0                        setErroAoEnviarEmails(true);
142  0                        setEmailsNaoEnviados("");
143  0                        for (int i = 0; i < listaNaoEnviados.size(); i++) {
144  0                            setEmailsNaoEnviados(getEmailsNaoEnviados() + listaNaoEnviados.get(i));
145                           }
146  0                        e.printStackTrace();
147  0                    }
148  0                    ex.printStackTrace();
149  2                }
150               }
151  1              if (listaNaoEnviados.size() == 0) {
152  1                  addSuccessMessage("E-mails enviados com sucesso!");
153  0              } else if (listaNaoEnviados.size() == emails.size()) {
154  0                  addMessage("Erro! Os e-mails não puderam ser enviados!");
155              } else {
156  0                  addMessage("Erro! Alguns e-mails não puderam ser enviados!");
157              }
158  0          } catch (Exception ex) {
159  0              ex.printStackTrace();
160  0              return null;
161  1          }
162  1          return null;
163         }
```

Figura 6.11 – Relatório de cobertura dos testes realizados de maneira informal

É importante ressaltar que as linhas não cobertas pelo teste informal eram de extrema importância para a avaliação da funcionalidade. Foi possível observar também, que estruturar o processo de teste dedicando um tempo maior para pensar nos possíveis cenários de teste para cada funcionalidade, permitiu alcançar uma cobertura maior no código desta funcionalidade já na primeira vez que os testes

foram realizados. Defeitos que não seriam detectados com a execução informal foram identificados com o processo de teste da PW-PlanTe.

A discrepância entre os resultados é grande também com relação ao tempo de execução do processo de teste. A atividade de teste, por meio de um processo formalizado como o proposto pela estratégia PW-PlanTe, tende a consumir mais recursos do que um processo de teste realizado de maneira informal. Entretanto, como esse estudo foi a primeira aplicação da estratégia na empresa, e o primeiro contato do desenvolvedor com o processo e com as ferramentas, a diferença de tempo entre a aplicação da PW-PlanTe e do processo informal, refletem o tempo gasto com a aplicação dos testes e com o aprendizado das novas atividades. Dessa forma, o processo estruturado com a seqüência de passos descritos anteriormente, naturalmente levou mais tempo que o processo informal. Faz-se necessário realizar outros estudos de caso para analisar o tempo de aplicação da estratégia considerando somente a realização das atividades de teste, sem considerar o tempo empregado no seu aprendizado.

6.4.2 Exemplo de aplicação 4 – Aplicação do processo de teste em sistema web já desenvolvido

Esse estudo foi realizado em um sistema de gerenciamento de páginas web desenvolvido pela empresa Linkway para o sistema web Toalhas São Carlos. Esse sistema compreende um portal web de uma indústria de toalhas, que necessita gerenciar o conteúdo inserido em seu *web site* tais como: contatos, vagas disponíveis na empresa, currículos cadastrados e representantes autorizados a vender o produto. Esse sistema, na época do estudo de caso, já se encontrava desenvolvido e em uso no ambiente da empresa.

Os resultados aqui apresentados foram obtidos por meio dos testes realizados somente no módulo de gerenciamento de representantes, no qual é possível listar todos os representantes cadastrados e realizar o cadastro de um novo representante. O objetivo desse estudo foi verificar se com o processo de teste da estratégia seria possível identificar defeitos não identificados com o teste ad-hoc.

Na falta de documentação sobre o sistema, baseou-se em conversas com desenvolvedores a fim de entender quais seriam os campos obrigatórios e a funcionalidade associada a esse módulo. Para a realização dos testes foram

inicialmente definidos os casos de teste a serem aplicados de forma manual (Etapa A). Foram estabelecidos para isso 16 casos de teste registrados na *FireScrum*, que se resumiam aos cenários descritos na Tabela 6.6.

Tabela 6.6 – Cenários de teste para o módulo "Gerenciamento de Representantes"

Cenário	Definição	Resultado Esperado
1	Verificar se são exibidos na tela de cadastro as seguintes áreas para preenchimento de informações: Informações Pessoais, Informações Profissionais, Informações do Endereço e Informações de Publicação.	Sucesso
2	Verificar se no "combo" da pergunta "Deseja aparecer na área publica do site" aparecem as duas opções: SIM e NÃO. Verificar se ao marcar SIM o representante será exibido na área pública do site	Sucesso
3	Verificar se no "combo" da pergunta "Deseja aparecer na área publica do site" ao marcar NÃO o representante não será exibido na área pública do site	Sucesso
4	Verificar se no "combo" do Estado são exibidos os 27 estados brasileiros	Sucesso
5	Criar um novo representante preenchendo todos os dados	Sucesso
6	Criar um novo representante preenchendo somente os dados obrigatórios	Sucesso
7	Criar um novo representante preenchendo todos os dados obrigatórios menos e-mail	Falha
8	Criar um novo representante preenchendo todos os dados obrigatórios menos Nome	Falha
9	Criar um novo representante preenchendo todos os dados obrigatórios menos Empresa	Falha
10	Criar um novo representante preenchendo todos os dados obrigatórios menos Telefone	Falha
11	Criar um novo representante preenchendo todos os dados obrigatórios menos Bairro	Falha
12	Criar um novo representante preenchendo todos os dados obrigatórios menos Cidade	Falha
13	Criar um novo representante preenchendo todos os dados obrigatórios menos Estado	Falha
14	Criar um novo representante preenchendo todos os dados obrigatórios menos País	Falha
15	Criar um novo representante preenchendo todos os dados obrigatórios menos Telefone Celular	Falha
16	Inserir quantidade de caracteres superior ao permitido no campo CNPJ	Falha

Na Etapa B, foram executados todos os casos de teste exibidos na Tabela 6.6, com o sistema já instrumentado pela ferramenta Cobertura, e diversos defeitos foram encontrados. Esses defeitos foram registrados na *FireScrum* (Etapa C) no módulo *Bug Tracking*, a cobertura foi verificada e os relatórios de cobertura gerados e analisados (Etapas D e E).

Os defeitos identificados com os casos de teste executados podem ser visualizados na Tabela 6.7.

Tabela 6.7 – Defeitos encontrados no módulo "Gerenciamento de Representantes"

Cenário	Definição	Resultado Esperado	Defeitos
1	Verificar se são exibidos na tela de cadastro as seguintes áreas para preenchimento de informações: Informações Pessoais, Informações Profissionais, Informações do Endereço e Informações de Publicação.	Sucesso	Na área Endereço, é exibido: "Informações Informações do Endereço"
3	Verificar se no "combo" da pergunta "Deseja aparecer na área publica do site" ao marcar NÃO o representante não será exibido na área pública do site	Sucesso	Mesmo marcando NÃO o representante inserido é exibido na área pública do site
4	Verificar se no "combo" do Estado são exibidos os 27 estados brasileiros	Sucesso	Falta o estado "Paraná"
5	Criar um novo representante preenchendo todos os dados	Sucesso	O campo Telefone Celular não é marcado como obrigatório
6	Criar um novo representante preenchendo somente os dados obrigatórios	Sucesso	O campo CPF só permite a inserção de 10 caracteres
15	Criar um novo representante preenchendo todos os dados obrigatórios menos Telefone Celular	Falha	O sistema gera uma exceção e não exibe nenhuma mensagem ao usuário
16	Inserir quantidade de caracteres superior ao permitido no campo CNPJ	Falha	O teste falha, entretanto não é exibida mensagem ao usuário somente a exceção.

Por meio dos relatórios de cobertura foi possível verificar o código, avaliando se as funcionalidades testadas foram realmente exercitadas. Na Figura 6.12, por exemplo, pode ser observado o trecho de código exercitado para o cenário de teste 4 (Tabela 6.6). Nesse cenário, deveria ser selecionado no "*combo box*" existente no cadastro, o estado do representante que está sendo cadastrado no momento. Nesse trecho de código é possível notar o defeito encontrado por esse caso de teste, que se refere a falta de exibição do estado Paraná como um dos 27 estados brasileiros.

```
167         public SelectItem[] getEstados() {
168
169  53         SelectItem[] items = new SelectItem[27];
170  53         int i = 0;
171  53         items[0] = new SelectItem("", "---");
172  53         items[1] = new SelectItem("AC", "Acre");
173  53         items[2] = new SelectItem("AL", "Alagoas");
174  53         items[3] = new SelectItem("AP", "Amapá");
175  53         items[4] = new SelectItem("AM", "Amazonas");
176  53         items[5] = new SelectItem("BA", "Bahia");
177  53         items[6] = new SelectItem("CE", "Ceará");
178  53         items[7] = new SelectItem("DF", "Distrito Federal");
179  53         items[8] = new SelectItem("ES", "Espírito Santo");
180  53         items[9] = new SelectItem("GO", "Goiás");
181  53         items[10] = new SelectItem("MA", "Maranhão");
182  53         items[11] = new SelectItem("MT", "Mato Grosso");
183  53         items[12] = new SelectItem("MS", "Mato Grosso do Sul");
184  53         items[13] = new SelectItem("MG", "Minas Gerais");
185  53         items[14] = new SelectItem("PA", "Pará");
186  53         items[15] = new SelectItem("PB", "Paraíba");
187  53         items[16] = new SelectItem("PE", "Pernambuco");
188  53         items[17] = new SelectItem("PI", "Piauí");
189  53         items[18] = new SelectItem("RJ", "Rio de Janeiro");
190  53         items[19] = new SelectItem("RN", "Rio Grande do Norte");
191  53         items[20] = new SelectItem("RS", "Rio Grande do Sul");
192  53         items[21] = new SelectItem("RO", "Rondônia");
193  53         items[22] = new SelectItem("RR", "Roraima");
194  53         items[23] = new SelectItem("SC", "Santa Catarina");
195  53         items[24] = new SelectItem("SP", "São Paulo");
196  53         items[25] = new SelectItem("SE", "Sergipe");
197  53         items[26] = new SelectItem("TO", "Tocantins");
198  53         return items;
199         }
```

Figura 6.12 – Relatório de cobertura dos testes realizados com o processo formalizado

Os outros defeitos identificados e apresentados na Tabela 6.7, não puderam ser visualizados nos relatórios de cobertura, pois devido à arquitetura de software utilizada, os trechos de código exercitados estavam nos arquivos *jsp* da aplicação que não podem ser cobertos com a ferramenta Cobertura. Entretanto, mesmo não sendo possível visualizar essa cobertura, alguns trechos de código *java* permitem visualizar evidências de que o código relativo à funcionalidade deve ter sido executado. Nas Figura 6.13 e Figura 6.14 é possível visualizar diferentes trechos de código que evidenciam, por exemplo, a execução do cadastro do representante. Na Figura 6.13 é exibido um trecho da classe relacionada à criação do usuário representante e na Figura 6.14 é exibido trecho da classe relacionada ao cadastro de usuários e endereço.

```
219     public String preparaCadstroClienteUsuarioAdmin() throws Exception {
220  4      cliente = clienteBO.setDadosClienteFromUsuario(usuario, cliente);
221  3      ecommerceEnderecoEntrega = clienteBO.setDadosEnderecoEntregaFromEnderecoUsuario(ecommerceEnderecoEntrega, enderecoUsuario);
222
223            try {
224  3             cliente = clienteBO.cadastrarNovoCliente(cliente, ecommerceEnderecoEntrega);
225  3             usuario = usuarioBO.cadastrarUsuario(usuario);
226  3             enderecoUsuario = usuarioBO.cadastrarEnderecoDoUsuario(enderecoUsuario, usuario);
227  0         } catch (Exception e) {
228  0             String msg = "Não foi possivel realizar o cadastro";
229  0             throw new Exception(msg, e);
230  3         }
231
232  3         return "SUCCESS";
233     }
```

Figura 6.13 – Relatório de cobertura da classe de criação de representante

```
174
175     public Usuario cadastrarUsuario(Usuario usuario) throws Exception {
176
177 3       EntityManager em = getEntityManager();
178
179         try {
180 3           em.getTransaction().begin();
181 3           usuario.setDataCriacao(new Date());
182 3           usuario.setDataAtualizacao(new Date());
183 3           em.persist(usuario);
184 3           em.getTransaction().commit();
185
186 0       } catch (Exception e) {
187 0           String msg = "Não foi possivel realizar o cadastro do Usuario";
188 0           throw new Exception(msg, e);
189 3       }
190 3       return usuario;
191
192     }
193
194     public EnderecoUsuario cadastrarEnderecoDoUsuario(EnderecoUsuario enderecoUsuario, Usuario usuario) throws Exception {
195 3       EntityManager em = getEntityManager();
196         try {
197
198 3           em.getTransaction().begin();
199 3           enderecoUsuario.setUsuario(usuario);
200 3           enderecoUsuario.setUsuarioId(usuario.getId());
201 3           em.persist(enderecoUsuario);
202 3           em.getTransaction().commit();
203
204 0       } catch (Exception e) {
205 0           String msg = "Não foi possivel atualizar o endereco do usuario";
206 0           throw new Exception(msg, e);
207
208 3       }
209 3       return enderecoUsuario;
210
211     }
```

Figura 6.14 – Relatório de cobertura da classe relacionada ao cadastro de usuário e de endereço de usuário

Com estes resultados é possível perceber, que mesmo com atividades simples de teste de software é possível identificar defeitos no software produzido. Como o sistema foi todo desenvolvido com a ausência da atividade de teste no processo de desenvolvimento os erros foram repassados para o ambiente do cliente o que acrescenta atividades de manutenção no momento em que esses defeitos sejam descobertos.

6.5 Considerações finais

Neste capítulo foram apresentados 4 estudos de caso dos quais dois deles exploraram a aplicação da estratégia PW-PlanTe no planejamento de software e os outros dois exemplificaram a aplicação do processo de teste da PW-PlanTe.

Como esses foram os primeiros estudos realizados, a avaliação da estratégia como um todo não pode ser realizada conforme havia sido planejado no inicio do desenvolvimento deste trabalho. Com a inserção de novas atividades, e por ser a

primeira vez que as equipes aplicavam a estratégia juntamente ao uso das ferramentas, nem todas as informações foram registradas nas ferramentas, principalmente relacionadas a teste. Dessa forma, não foi possível obter uma análise mais significativa entre o tempo gasto em desenvolvimento e em teste, de maneira que seja possível planejar melhor essas duas atividades no processo de desenvolvimento.

Por modificar a rotina de trabalho da empresa, a resistência à realização das atividades propostas, mesmo que selecionadas com o objetivo de acrescentar pouco esforço ao processo como um todo, foi grande. Tais problemas mostram a dificuldade em realizar a implantação de melhoria de processo em empresas e também a dificuldade em realizar estudos em um ambiente real de desenvolvimento, no qual o objeto de estudo está em constante modificação.

Entretanto mesmo com essas limitações foi possível observar com os estudos de caso que o uso da estratégia é viável e que, se corretamente aplicada pode vir a auxiliar no planejamento e também na verificação da qualidade do software desenvolvido.

Como pôde ser observado nos dois primeiros estudos de caso, as atividades de planejamento permitem a equipe planejar e reavaliar o planejamento efetuado de maneira que grandes discrepâncias na produtividade da equipe sejam identificadas em tempo hábil de corrigi-las. E a aplicação do processo de teste, mostrada nos dois últimos estudos de caso, permitiu observar que a introdução de atividades de teste formalizadas permite a captura de defeitos antes que o sistema seja entregue ao cliente, agregando qualidade ao produto desenvolvido.

No próximo capítulo serão apresentadas as conclusões obtidas com o trabalho realizado, suas contribuições e limitações e possíveis trabalhos futuros.

Capítulo 7

CONCLUSÕES

Neste trabalho foi apresentada a Estratégia PW-PlanTe, que combina atividades de planejamento e teste com o objetivo de auxiliar na melhoria da qualidade do software desenvolvido e no planejamento e gerenciamento do software.

A Estratégia PW-PlanTe foi definida com base em um processo constante de observação e evolução, por meio do qual foram realizadas melhorias na Estratégia $PCU|_{PSP}$, a qual dá suporte ao planejamento e o acompanhamento do planejamento de software, com o apoio da técnica de estimativa PCU (Pontos por Caso de Uso) e do processo PSP (*Personal Software Process*).

A estratégia PW-PlanTe foi definida no contexto da mesma empresa para a qual foi definida a estratégia $PCU|_{PSP}$, Dando continuidade ao constante objetivo da empresa em implantar melhorias em seu processo de desenvolvimento, o principal alvo de melhoria a ser tratado era o processo de teste.

No entanto, como a empresa passou a adotar o framework ágil Scrum, e como a $PCU|_{PSP}$ era muito aderente a ele, fez-se também uma melhoria nessa estratégia, no que diz respeito à atividade de planejamento. Essa melhoria correspondeu a considerar como suporte ao planejamento, outras técnicas de estimativa, além do PCU, técnicas essas mais utilizadas no contexto dos métodos ágeis. Essa evolução permite agora que a empresa trabalhe com a unidade de trabalho PW – Piece of Work, ao invés de casos de uso. O PW corresponde a uma "porção de trabalho" que cabe em uma iteração; isso tornou a estratégia mais genérica e mais fácil de ser adotada por qualquer empresa.

A PW-PlanTe conservou o controle do planejamento já existente na $PCU|_{PSP}$, realizado por meio do constante ajuste da quantidade de trabalho a ser desenvolvida

em cada iteração. Esse ajuste é realizado com base nas medidas de tempo coletadas com o uso do PSP, que são utilizadas no cálculo do Nível de Esforço (NDE), que retrata a produtividade da equipe em cada iteração. Com isso, o gerente pode acompanhar o desenvolvimento do trabalho e saber se existe a chance de ocorrer atrasos na entrega do software. Esse acompanhamento é de grande importância no contexto dos métodos ágeis no qual os prazos são mais curtos e as entregas devem ser realizadas de forma rápida.

No que diz respeito à atividade de teste, a PW-PlanTe definiu um processo de teste com o objetivo de causar o menor impacto possível, permitindo a sua aplicação em um ambiente de empresas de pequeno porte, no qual as equipes são geralmente pequenas e conta-se com poucos recursos.

Dessa forma, foram selecionadas atividades de teste que pudessem melhorar a qualidade do produto de software desenvolvido, com base em critérios de teste mais simples e que são considerados como o mínimo de qualidade desejável em uma atividade de teste, a saber, os critérios todos-nós e todos-ramos. Para tanto as atividades de teste que compõem o processo correspondem à criação de cenários de teste que exercitem o software desenvolvido com o objetivo de revelar defeitos; à execução desses cenários; a análise da cobertura; e o registro de defeitos.

Para dar suporte a todo processo, foi realizada uma busca por ferramentas livres, que pudessem apoiar a realização dessas atividades, de forma a não gerar custos para a empresa. Essa busca resultou na seleção das ferramentas FireScrum, Selenium e Cobertura. A FireScrum foi selecionada devido ao fato de permitir o planejamento e o gerenciamento do trabalho da equipe no contexto de métodos ágeis, e também por permitir o registro e planejamento dos casos de teste e dos defeitos encontrados. A Selenium, por sua vez, foi selecionada por permitir a execução automatizada de teste, considerando que, em alguns momentos, a execução dos testes repetidamente se faz necessária, como no teste de desempenho. E, por fim, a ferramenta Cobertura, que permite verificar o cumprimento dos critérios de teste. Esse conjunto de ferramentas permitiu estabelecer um processo de teste com um custo reduzido e que pode ser aplicado em qualquer empresa que desenvolva aplicativos para *web* em Java.

A parceria com duas empresas de desenvolvimento web de pequeno porte, a Linkway e NBS, permitiu a realização de atividades de observação e definição da

PW-PlanTe, bem como a realização de estudos de caso que permitiram avaliar a estratégia proposta.

Nos estudos de caso realizados foi possível observar como a estratégia pode ser aplicada no processo de desenvolvimento no contexto de planejamento e teste de software. Ressalta-se que, pelo fato da estratégia estar sendo definida e implantada ao mesmo tempo e que isso implica em uma mudança cultural na empresa, a análise desses estudos ficou prejudicada. Como a estratégia ainda não se tornou rotina no dia a dia da empresa, houve algumas adaptações durante o seu uso, o que não permitiu avaliar com maior segurança as contribuições que podem ser obtidas do seu uso constante.

No primeiro e segundo estudos de caso explorou-se a atividade de planejamento e controle da PW-PlanTe em dois projetos de sistemas web que estavam ainda em desenvolvimento: o CondLink e o Consulta Pública. Nesses estudos os resultados encontrados mostraram que, para realizar corretamente a aplicação da estratégia, é necessário que a equipe amadureça a rotina de trabalho proposta pela PW-PlanTe. Foi possível observar que, um fator que influenciou nos resultados, foi o fato das equipes ainda não conseguirem realizar a correta atribuição de pontos, ou seja, a correta caracterização da complexidade dos itens de trabalho. É necessário que a equipe, com o decorrer do tempo, consiga observar os resultados do seu trabalho e formar uma base de dados histórica que possa ser usada para estimar outros projetos. Com isso, será possível ao atribuir a complexidade do trabalho, levar em consideração essas informações históricas permitindo maior acerto nas estimativas. A variação do NDE entre os dois estudos de caso confirma o fato de que a produtividade é uma medida que deve ser coletada individualmente e calculada para cada equipe, pois o nível de conhecimento técnico e de rendimento varia de indivíduo a indivíduo.

No terceiro e quarto estudos de caso explorou-se a aplicação das atividades de teste da PW-PlanTe em dois projetos de sistemas web: o Consulta Pública e o Toalhas São Carlos. O Consulta Pública ainda estava em desenvolvimento e o processo de teste foi aplicado com o objetivo de compará-lo com a forma ad-hoc que era adotada pela empresa. O Toalhas São Carlos é um sistema web que já estava desenvolvido e o processo de teste foi utilizado com o objetivo de investigar se outros defeitos seriam encontrados com sua aplicação, que não o foram com o teste ad-hoc que já havia sido aplicado quando de seu desenvolvimento.

Nesses dois últimos estudos de caso foi possível observar que, com a introdução de atividades de teste formalizadas, a chance dos defeitos inseridos no processo de desenvolvimento chegarem ao ambiente do cliente fica reduzida. Pode-se perceber isso pela diferença da quantidade de defeitos encontrados na realização dos dois estudos de caso. Nota-se que o simples fato de solicitar que os cenários de teste fossem pensados antes da execução dos testes, já possibilitou que situações não exploradas no processo informal, fossem identificadas, provocando, portanto a criação de mais casos de teste e aumentando, conseqüentemente, a cobertura do código. Com uma quantidade maior de cenários testados, foi possível exercitar mais linhas de código e identificar mais defeitos que não foram encontrados com a execução dos testes ad-hoc. Essa identificação de defeitos, ainda no ciclo de desenvolvimento, permite que as correções sejam realizadas antes do sistema estar em uso no ambiente do cliente. Isso reduz a quantidade de retrabalho para a correção dos defeitos, quando comparada com o sistema em fase operacional.

A aplicação das atividades de planejamento, acompanhadas pelo controle constante do tempo gasto individualmente, permite que a equipe adquira um aprendizado maior quanto ao tempo consumido e passem a caracterizar melhor a complexidade do trabalho a ser desenvolvido. Além disso, a aplicação da estratégia auxilia na identificação da real produtividade, o que permite estimar o trabalho que deverá ser realizado de forma mais ajustada.

O gerente deve atuar durante o processo de desenvolvimento como um treinador, tornando visível à equipe o progresso do trabalho realizado, acompanhando a descoberta e resolução de defeitos, alertando para a baixa produtividade e incentivando o Time a melhorar cada vez mais seus processos.

Com os resultados obtidos com a utilização do processo de teste foi possível observar que com a aplicação das atividades propostas e das ferramentas selecionadas, a empresa de desenvolvimento pode sair de um estado em que tais atividades são realizadas de forma caótica a um estado mais formalizado. Assim, as atividades de garantia da qualidade se tornam mais visíveis a todos os envolvidos e influenciam na qualidade do produto final.

7.1 Contribuições e Limitações

As principais contribuições deste trabalho são:

- Generalização da Estratégia $PCU|_{PSP}$ quanto aos métodos de estimativa empregados, possibilitando que a estratégia possa ser utilizada em diferentes contextos de desenvolvimento;
- Estabelecimento de uma medida de produtividade que deve ser obtida em cada equipe, de forma que seja possível mensurar a capacidade de trabalho que pode ser alocada nas iterações de desenvolvimento;
- Definição de um processo de teste que apóia a realização de atividades de teste com baixo custo no ambiente de empresas de pequeno porte;
- Aproximação maior da estratégia ao framework ágil Scrum e aos princípios ágeis;
- Avaliação da Estratégia por meio de estudos de caso em ambiente real, no qual ela também foi definida.

Limitações:

- Por se tratar de uma pequena empresa, os estudos foram realizados em equipes com somente dois membros do time de desenvolvimento – um desenvolvedor e um testador. Assim, não foi possível verificar o funcionamento da estratégia em equipes com mais de um desenvolvedor ou testador trabalhando em paralelo na mesma equipe.
- A mudança cultural foi um fator limitante à aplicação da estratégia como um todo, pois, quando o processo de teste deveria ser aplicado juntamente com as atividades de planejamento, a equipe passou a se preocupar somente em registrar o tempo para realização das atividades de planejamento, negligenciando o registro das atividades de teste da forma como a Estratégia propõe.

7.2 Trabalhos Futuros

Dentre as atividades que podem ser realizadas como continuidade deste trabalho podem-se citar:

1. Aplicar a PW-PlanTe em projetos de equipes com mais do que um desenvolvedor, com o objetivo de examinar a estratégia em equipes maiores;
2. Amadurecer a estratégia com a inserção de outros tipos de atividade de garantia da qualidade tais como testes unitários e inspeções de software;
3. Realizar estudos de caso comparando as atividades de planejamento da estratégia com diferentes tipos de métodos de estimativa, verificando se para o mesmo projeto, com a aplicação de métodos diferentes existiriam variações nos resultados encontrados;
4. Verificar a conformidade da estratégia como facilitadora da implantação de modelos de qualidade como o CMMI e o MPS.BR;
5. Realizar experimentos controlados com o objetivo de medir os benefícios da aplicação da estratégia em uma equipe com desenvolvedores e testadores experientes.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAMSSON, P. et al. **Agile software development methods Review and analysis**. Relatório Técnico. Finlândia:VIT Publications. 2002.

ALBRECHT, A. Measuring application development productivity. In: SHARE/GUIDE IBM APPLICATION DEVELOPMENT SYMPOSIUM, 14-17 de Out. 1979, Monterey, California. **Proceedings...** Monterey: IBM Corporation, 1979, pp. 83--92.

AMBLER, S. W. **Agile Testing and Quality Strategies:** Discipline Over Rhetoric. Disponível em: http://www.ambysoft.com/essays/agileLifecycle.html. Acesso em: 16 janeiro de 2010.

_____. **The Agile System Development Life Cycle (SDLC)**. Disponível em: http://www.ambysoft.com/essays/agileLifecycle.html. Acesso em: 16 janeiro de 2010.

AMBYSOFT. **Test Driven Development (TDD) Survey Results:** October 2008. Disponível em: http://www.ambysoft.com/surveys/tdd2008.html. Acesso em: 16 de janeiro de 2010.

ANDERSSON, C.; RUNESON, P., Verification and Validation in industry - a qualitative survey on the state of practice, In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING (ISESE'02), 3-4 de Out. 2002, Nara, Japão. **Proceedings…** Washington: IEEE Computer Society, 2002, p.37.

BARBOSA, E. *et al.* Introdução ao teste de software. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, XIV, 4-6 de Out. 2000, João Pessoa, Paraíba. **Anais…** João Pessoa: SBES'2000, 2000, p 330–378.

BASILI, V. *et al.* Packaging researcher experience to assist replication of experiments. In: INTERNATIONAL SOFTWARE ENGINEERING RESEARCH NETWORK MEETING - ISERN. Sydney, Australia, 1996a. **Proceedings...** Sydney: ISERN 1996. p.

BECK, K. **Programação extrema explicada:** acolha as mudanças. 1 ed. Porto Alegre: Bookman, 2004. 182 p.

BEGEL, A; NAGAPPAN, N. Usage and perceptions of agile software development in an industrial context: An Exploratory Study, In: FIRST INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. 20-21 de Set. 2007, Madrid, Espanha. **Proceedings…**Washington: IEEE Computer Society, 2007, p. 255-264.

BESSON, F. M.; BEDER, D. M; CHAIM, M. L. **Um conjunto de ferramentas para modelagem de casos de teste de aceitação de aplicações web**. In: XVI SESSÃO

DE FERRAMENTAS DO SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE – SBES. Fortaleza, Brasil, 5-9 de Out. 2009.

BLACK, R. **Managing the testing process:** practical tools and techniques for managing hardware and software testing. 2 ed. Canada: Wiley Publishing, 2002. 500 p.

BOEHM, B. Get ready for agile methods, with care. **IEEE Computer**, v. 35, n. 1, p. 64 – 69, 2002.

CANÓS, J.H.; LETELIER, P.; PENADÉS, M.C. **Metodologias Ágiles em el Desarrollo de Software**. 2003. Disponível em: <http://www.willydev.net/descargas/prev/TodoAgil.Pdf>. Acesso em 08 jan. 2010.

CAVALCANTI, E; MACIEL, T. M. M.; ALBUQUERQUE, J. **Ferramenta Open-Source para Apoio ao Uso do Scrum por Equipes Distribuídas**. In: Workshop de Desenvolvimento Distribuído de Software, Fortaleza – CE, 2009.

CMMI. Capability Maturity Model Integration Version 1.2. (CMMI-SE/SW, V1.2 – Continuous Representation), **SEI Technical Report CMU/SEI-2006-TR-001**. 2006.

COBERTURA. **Cobertura.** Disponível em: http://cobertura.sourceforge.net/. Acesso em: 13 jan. 2010.

COCKBURN, A. **Agile Software development**. Boston: Addison Wesley, 2002. (The Agile Software Development Series). 304 p.

COHN, M. **Agile estimating and planning**. New Jersey: Prentice Hall PTR, 2005. 368 p.

CORAM, M.; BOHNER, S. The impact of agile methods on software project management. In IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS), 12. 2005, Greenbelt. **Proceedings…**Washington: IEEE Computer Society, 4-7 de Abr. 2005. pp. 363 -370.

DANUBE. **Scrum Tools - ScrumWorks Pro & ScrumWorks Basic**. Disponível em: http://www.danube.com/scrumworks. Acesso em: 13 de jan. de 2010.

DASHBOARD. **The Software Process Dashboard Project Team**. Acesso em: <http://processdash.sourceforge.net>. Acesso em: 13 jan. 2010.

DSDM. **DSDM public version 4.2 manual**. Disponível em: http://www.dsdm.org/version4/2/public/. Acesso em: 22 de jan. 2010.

HACKYSTAT. **Hackystat Project**. Disponível em: http://www.hackystat.org. Acesso em: 13 jan. 2010.

HEDBERG, H; IISAKKA, J. Technical Reviews in Agile Development: Case Mobile-DTM. In: International Conference on Quality Software (QSIC'06), 6, 2006, Beijing-China. **Proceedings…**Beijing: IEEE Computer Society, 2006. P. 347-353.

HIGHSMITH, J. **Agile software development ecosystems**. Boston: Addison-Wesley, 2002. 448 p.

HOLMES, A.; KELLOG, K. Automating Functional Tests Using Selenium. In: Agile Conference, 2006, Minneapolis-Minnesota. **Proceedings…**Washington: IEEE Computer Society, 23-28 de Jul. 2006. p. 270-275.

HUMPHREY, W. **A Discipline for Software Engineering.** Pittsburgh: Addison Wesley, 1995, 816p.

ISO. **NBR ISO/IEC 12207:1995**. Tecnologia da Informação – processos de ciclo de vida de software, 1995.

ISO. **ISO/IEC TR 15504:1998**(E). Information Technology - Software Process Assessments.In: International Organization for Standardization: Geneva. Partes 1-9, 1998.

JOHNSON, P. *et al*. Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined. In: INTERNATIONAL CONFERENCE OF SOFTWARE ENGINEERING, 25, 2003, Portland, Oregon. **Proceedings…** Portland: IEEE Computer Society, 2003, p. 641-646.

KARNER, G. **Use Case Points - Resource Estimation for Objectory Projects. Objective Systems SF AB**, University of Linköping, Suécia, 1993.

KNIBERG, H. **Scrum and XP from the Trenches** - How we do use Scrum. Disponível em: http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf. Acesso em: 11 jan. 2010.

LEWIS, W; VEERAPILLAI,G. Quality Assurance Framework. In: _____. **Software testing and continuous quality improvement**, 2ª Edição. Florida: CRC Press, 2004, p.5-27.

MAHER, P. Weaving Agile Software Development Techniques into a Traditional Computer Science Curriculum. In: International Conference on Information Technology: New Generations, 6, 2009, Las Vegas. **Proceedings…**Washington: IEEE Computer Society, 27-29 de Abr., 2009. p.1687-1688.

MALDONADO, J. **Critérios Potenciais Usos**: uma contribuição ao teste estrutural de software. 1991. 247 f. Tese (Doutorado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica, UNICAMP, Campinas, 1991.

MALDONADO, J; FABBRI, S. Verificação e validação de software. In: ROCHA, A; MALDONADO, J; WEBER, K. **Qualidade de Software: Teoria e Prática**. São Paulo: Prentice Hall, 2001a, p.66-73.

MALDONADO, J; FABBRI, S. Teste de Software. In: ROCHA, A; MALDONADO, J; WEBER, K. **Qualidade de Software: Teoria e Prática**. São Paulo: Prentice Hall, 2001b, p 73-84.

MANIFESTO. **Manifesto for Agile Software Development**. (2001). Retrieved November 5, 2008, from http://agilemanifesto.org/

MANTIS. **Mantis Bug Tracker**. Disponível em: http://www.mantisbt.org/. Acesso em: 13 de jan. de 2010.

MONTEBELO,R. **Identificando Dificuldades e Benefícios do Uso do PSP Apoiado por Ferramentas de 3ª Geração**. 2008. 142 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2008.

MPSBR. **Melhoria de Processo do Software Brasileiro – Guia Geral (Versão 1.2).** Associação para promoção da excelência do software brasileiro. 2007.

NETO, A. C. *et al*. Caracterização do Estado da Prática das Atividades de Teste em um Cenário de Desenvolvimento de Software Brasileiro. In: Simpósio Brasileiro de Qualidade de Software – SBQS´2006, 5, 2006, Vila Velha – ES. Simpósio Brasileiro de Qualidade de Software, 2006.

OPELT, K; BEESON, T. 2008, Agile Teams Require Agile QA: How to make it work, an experience report, Agile 2008 Conference, 2008, Toronto. **Proceedings…**Washington: IEEE Computer Society, 2008. p. 229-232.

PALMER, S. R.; Felsing, J. M. **A Practical Guide to Feature-Driven Development**. New Jersey: Prentice-Hall. 304 p.

PAULK, M *et al*. Capability Maturity Model for Software, Version 1.1. **SEI Technical Report CMU/SEI-93-TR-024**, 1993.

POTTS, C. Software-Engineering Research Revisited. **IEEE Software**, Los Alamitos – CA, v. 10, n. 5, p. 19-28, set. 1993.

PRESSMAN, R. **Engenharia de Software.** 6ª Edição. São Paulo: McGraw-Hill, 2006. 880p.

REHESAAR, H.; BEAMES, E. Project Plans and Time Budgets in Information Systems Projects. IN: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING EDUCATION & PRACTICE, 1998, Dunedin, Nova Zelândia. **Proceedings...** Nova Zelândia: IEEE Computer Society, 1998, p. 120-124.

SANCHEZ, A. **PCU$|_{PSP}$ Uma Estratégia para Ajustar Pontos por Caso de Uso Baseada em PSP**. 2008. 133 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2008.

SANDE, D. M *et al.*. PW-Plan: a strategy to support iteration-based software Planning. In: International Conference on Enterprise Information Systems (ICEIS 2010), 12, 2010, Portugal. **Proceedings...**Ilha da Madeira: 2010.

SCHWABER, K. **Agile project management with Scrum**. Redmond USA: Microsoft Press, 2004. 192 p.

SELENIUM. **Selenium web application testing system**. Disponível em: http://seleniumhq.org/. Acesso em: 13 de jan. de 2010.

STANDISH. **The chaos report**. 1995. Disponível em: http://www.standishgroup.com. Acesso em: 12 de março de 2010.

_____. **Extreme Chaos**. 2001. Disponível em: http://www.standishgroup.com. Acesso em: 12 de mar. de 2010.

TESTLINK. **Test management tool = TestLink**. Disponível em: http://www.teamst.org/. Acesso em: 13 de jan. de 2010.

TIAN, J. **Testing: Concepts, Issues, and Techniques**. In: _____. Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement. New Jersey: John Wiley & Sons, 2005, p. 67-84.

VICENZI, A. M. R. *et al*. **Coverage testing of Java programs and components**. Science of Computer Programming, Elsevier North-Holland, Amsterdam, v. 56, n. 1-2, p. 211-230, abr. 2005.

Anexo A

COMO UTILIZAR A FERRAMENTA COBERTURA

A ferramenta Cobertura pode ser utilizada de forma integrada com a ferramenta para automatização da construção de software o Apache Ant ou diretamente por linha de comando.

Para utilizar a ferramenta por meio de linha de comando é necessário seguir uma seqüência de ações tais como instrumentar o código, executar testes e gerar relatórios de cobertura.

Para entender melhor o funcionamento a seguir estão descritos passos detalhados de como usar a ferramenta.

Recomenda-se que caso esteja sendo utilizada uma instância do Tomcat juntamente com a IDE de desenvolvimento, utilizar outra instância do Tomcat. Caso ocorra algum problema durante a execução dos passos abaixo apagar o contexto guardado na pasta C:\apache-tomcat-versao\work.

1. Preparar código para instrumentação

1.1. Para que seja possível instrumentar o código deve ser gerado um arquivo compilado da aplicação ".war".

1.2. Após gerar o .war da aplicação copiar o arquivo .war para a pasta \apache-tomcat-versao\webapps

1.3. Inserir o jar do cobertura (cobertura.jar) dentro da pasta WEB-INF\lib do .war.

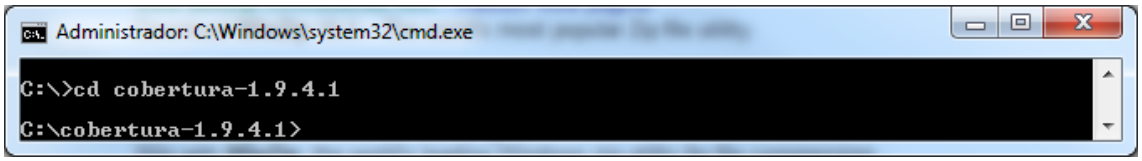
Para executar esta ação pode ser utilizado um software compactador de arquivos como por exemplo o WinZip. Abra o arquivo .war utilizando o

WinZip, navegue até a pasta WEB-INF\lib e arraste o arquivo cobertura.jar para esta pasta.

1.4. Novamente usando o WinZip, extrair do .war as classes da aplicação armazenadas na pasta WEB-INF\classes

2. Instrumentar código

2.1. Abrir o prompt de comando e entrar no diretório do Cobertura



2.2. Inserir a instrução elaborada segundo o seguinte formato:

```
cobertura-instrument.bat [--basedir dir] [--datafile file] [--destination dir] [--ignore regex] classes [...]
```

Exemplo:

```
cobertura-instrument.bat --datafile=C:\apache-tomcat-versao\bin\cobertura.ser C:\diretorio-usado-passo-1.4\classes
```

Este comando identifica que as classes presentes no diretório C:\diretorio-usado-no-passo-1.4\classes foram instrumentadas e as informações serializadas sobre tais classes foram armazenadas no arquivo cobertura.ser armazenado no diretório C:\apache-tomcat-versao\bin\cobertura.ser.

2.3. Usando o WinZip inserir classes (já instrumentadas) no .war no diretório WEB-INF\classes

3. Executar testes

3.1. Executar o tomcat:

3.1.1. Por meio do prompt de comando entrar em C:\apache-tomcat-versao\bin

3.1.2. Executar o comando

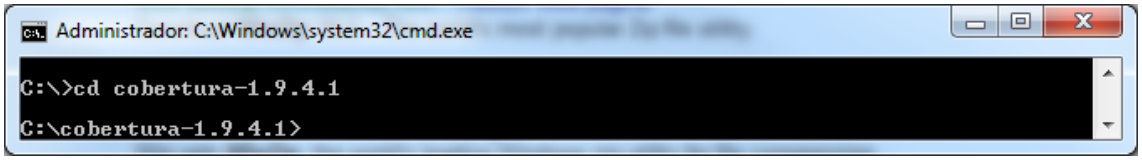
```
java -jar -Dnet.sourceforge.cobertura.datafile=C:\apache-tomcat-versao\bin\cobertura.ser bootstrap.jar
```

3.2. Executar testes na aplicação de maneira manual ou utilizando uma ferramenta para execução de testes como por exemplo o Selenium

3.3. Parar servidor após testes finalizados.

4. Gerar relatórios de cobertura

4.1. Por meio do prompt de comando entrar no diretório do cobertura:



4.2. Inserir a instrução elaborada segundo o seguinte formato:

```
cobertura-report.bat [--datafile file] [--destination dir]
[--format (html|xml)] [--encoding encoding] source code directory
[...] [--basedir dir file underneath basedir ...]
```

Exemplo:

```
cobertura-report.bat --datafile C:\apache-tomcat-
versao\bin\cobertura.ser --destination C:\diretorio-
relatorios_de_testes C:\diretorio-arquivos-java\src\java
```

Este comando determina que as informações sobre a execução dos testes armazenadas no arquivo cobertura.ser sejam utilizadas, juntamente com o código fonte da aplicação armazenado em C:\diretorio-arquivos-java\src\java, para geração dos relatórios de cobertura. Tais relatórios serão armazenados no diretório C:\diretorio-relatorios_de_testes.

Mais informações sobre os comandos da ferramenta cobertura podem ser encontradas na página da ferramenta no endereço http://cobertura.sourceforge.net.