UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

MARCELO MOTA MANHÃES

Classificação e Resolução de Defeitos em Manutenção de Software utilizando ODC e Histórico de Soluções

DISSERTAÇÃO DE MESTRADO

CURITIBA
2014

MARCELO MOTA MANHÃES

Classificação e Resolução de Defeitos em Manutenção de Software utilizando ODC e Histórico de Soluções

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Área de concentração: Engenharia de Sistemas Computacionais

Orientador: Laudelino Cordeiro Bastos
Coorientadora: Maria Claudia F. P. Emer

CURITIBA
2014

Resumo

Nos dias atuais, com o aumento da demanda de serviços de suporte, como, por exemplo, no campo da alta disponibilidade e do desempenho para o cliente e da demanda de custos mais baixos para as empresas de manutenção e hospedagem de software, a resolução de incidentes de software em um tempo menor junto ao cliente e a prevenção de defeitos tornaram-se um tópicos fundamentais. Além disso, a atividade de manutenção de software é uma das fases que consome mais tempo, esforço e consequentemente custo no ciclo de desenvolvimento de software. Balancear eficiência e custo torna-se um desafio para qualquer empresa de suporte em manutenção de software. A abordagem utilizada nesta pesquisa estabelece um processo para classificar defeitos e delinear um conjunto de melhores soluções para os defeitos classificados a partir do histórico de defeitos e a base de conhecimento do cliente. Esta classificação também separa complexidade de problemas para serem gerenciados pelo time de suporte mais adequado. O método base utilizado é o ODC (Classificação Ortogonal de Defeitos) e extensões voltadas ao suporte de software são propostas e utilizadas. Por meio desta pesquisa, é possível verificar se a classificação e associação de soluções podem acarretar em uma redução no tempo de atendimento dos incidentes de suporte. Foi observado em quatro amostras de dois clientes diferentes (X e Y) que utilizando a classificação dos defeitos, direcionamento correto aos times de suporte e agrupamento de soluções, promoveu uma redução no tempo de atendimento em 70% dos incidentes de suporte no cliente Y e 92,5% dos incidentes de suporte para o cliente X. A redução de tempo foi obtida pela redução no número de transferências entre os times de suporte e a redução de incidentes. O processo apresentado é incremental, pois é baseado no aumento das informações históricas e na eficácia das soluções propostas. Este método de soluções pode favorecer a redução dos recursos necessários para suportar sistemas computacionais em provedores de serviço.

Palavras-chave: Classificação de Defeitos, ODC, Prevenção de Defeitos, Histórico de Soluções, Manutenção de Software.

Abstract

Nowadays with the increasing of demand in support services such as high availability and performance faced by customers and low cost operations to software maintenance and hosting enterprises, the resolution of software incidents in less time and defect prevention turned a key point. Moreover, the software maintenance is one most time consuming and effort demanding and by consequence cost in software development life cycle. Balancing effectiveness and costs turn a challenge to any enterprise that manages support in software maintenance. The approach used on this research defines a process to classify defects and boundary a set of best solutions associated to these classes from defects history and customer knowledge base. Also this classification separates different problem complexities that can be handled to different support teams. The base method used is the ODC (Orthogonal Defect Classification). With this research is possible to verify that the classification and solutions associations with problem class can undertake a time reduction in incidents resolution in software maintenance. It was verified using four service provider samples in two different customers (X and Y) that classification of defects, correct support team redirection and best solution grouping helps on reduction of incidents resolution time between 70% of incidents to customer Y and 92,5 % of incidents to customer X. The reduction was reached with reduction in number of transfers between supporting teams and incidents number reduction. The process is incremental because it is unfolded from history information and from effectiveness into solutions purposed. This process can leverage human resources nedded to support computational systems in service providers.

Keywords: Defect Classification, ODC, Defect Prevention, Solutions History, Software Maintenance.

Sumário

Lista de Figuras

Lista de Quadros

Lista de Abreviações e siglas

ODC: *Orthogonal Defect Classification* (Classificação Ortogonal de Defeitos).

OPC: *Orthogonal Problem Classification* (Classificação Ortogonal de Problemas).

SLA: *Service Level Agreement* (Acordo de Nível de Serviço).

SLO: *Service Level Objectives* (Objetivos de Nível de Serviço).

MTTR: *Mean Time to Resolve* (Tempo Médio de Resolução do Problema).

ITIL: *Information Technology Service Library* (Blioteca de Serviços de TI).

CMM: *Capability Maturity Model Integration* (Modelo de Integração de Capacidade e Maturidade).

FFDA: *First Failure Data Analysis* (Análise de Dados da Primeira Falha)

TCO: *Total Cost of Ownership* (Custo Total do Domínio)

OLAP: *On-line Analytical Processing* (Processamento Analítico em Tempo-real)

1 INTRODUÇÃO

Em anos recentes, a indústria de serviços de manutenção e hospedagem de software tem-se deparado com uma pressão contínua pela qualidade dos serviços, como a melhora no atendimento ao cliente, a restauração rápida para a indisponibilidade do serviço e a diminuição do tempo de resposta do serviço prestado. Da mesma forma, essas empresas de manutenção de software necessitam de uma redução de custos para sobreviver ao mercado competitivo. Esse conflito, entre a melhoria da qualidade e a redução do custo, leva a indústria de software a explorar métodos inovadores para gerenciar o negócio. Métricas comuns para medir a qualidade do serviço, como a disponibilidade do equipamento, o tempo para resolver incidentes, ou *Mean Time to Resolve* (MTTR), são usadas regularmente como parte do gerenciamento operacional padrão nos serviços de manutenção de software (DIAO, 2011).

O primeiro problema que pode ser identificado é a qualidade dos serviços prestados. Isto é o resultado da lacuna entre os modelos estatísticos (quantitativos) e o modelo de análise de causa raiz dos defeitos. Existe uma grande lacuna entre os dados estatísticos dos defeitos e a resolução dos mesmos defeitos em termos de prevenção de defeitos. Na Figura 1, pode-se observar que existe um campo de pesquisa muito grande no espectro de análise entre causa e efeito. Preencher este espectro permite que esses defeitos sejam corrigidos de forma mais eficaz, menos custosa e menos orientada a cada problema individual (CHILLAREGE, 1994).

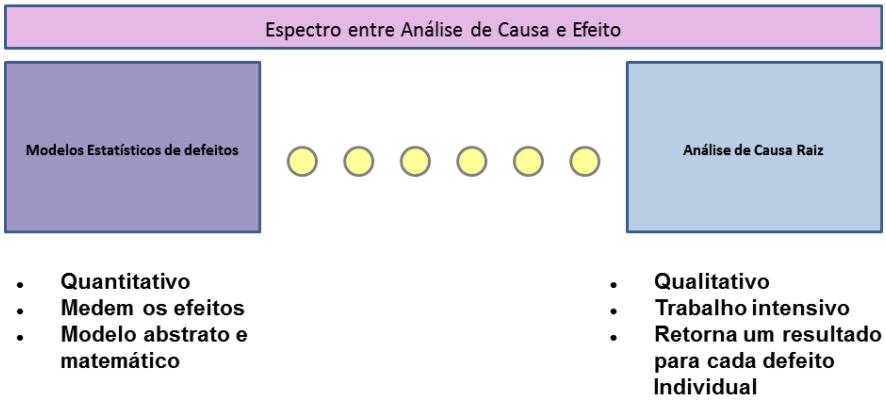

Figura 1 - O espectro de análise de causa e efeito nos dados estatísticos de um lado e o trabalho de análise de causa raiz de outro. Adaptado de Chillarege (1994).

Seguindo o espectro entre causa e efeito a qualidade orienta-se pela satisfação do cliente. Buckley e Chillarege (1995) apontam que a ordem relativa de influência na satisfação do cliente, a partir de quatro medidas chave do serviço de suporte, que são usualmente monitoradas, indica primeiramente a resolução dos defeitos seguida do número de problemas. Estas duas medidas são seguidas, por sua vez, pelo número de defeitos e pelo período de resolução. Estes dados reforçam que a prevenção de defeitos e a eficácia na solução dos problemas têm mais influência na satisfação do cliente do que o número de problemas em si.

O segundo problema a ser considerado é a complexidade dos serviços de manutenção e hospedagem de software. Um provedor de serviços, tanto localmente como em outro país, oferecendo serviços para clientes que estão localizados globalmente é um cenário cada vez mais presente com o advento e utilização da computação em nuvem. Um cliente pode ter diversas necessidades, como gerenciamento de rede, suporte a banco de dados, necessidade de serviços de backup, serviços de restauração de dados, hospedagem de sua aplicação, somente citando alguns. O cliente analisa o portfólio do fornecedor e o contrata por um ou mais serviços. A infra-estrutura para suportar estes serviços, isto é, servidores, estrutura de redes, aplicações e processos de negócios podem ser de propriedade do cliente como podem ser localizados no provedor de serviços (DIAO, 2011). Essa complexidade requer times de suporte cada vez mais capacitados a atuar nesse tipo de infraestrutura.

O terceiro problema abordado é o custo da manutenção de software, que é uma das atividades que mais consomem tempo, esforço e, consequentemente, dinheiro no ciclo de desenvolvimento de software. Gerentes de manutenção procuram métodos e a equipe técnica procura por métodos e ferramentas que suportem os cronogramas e a execução de diferentes tarefas de manutenção, como mudanças de software, de forma que sejam evitados incidentes e problemas associados (REFORMAT; IGBIDE, 2005). Estudos, como em Buckley e Chillarege (1995), apontam que, para cada 1 dólar investido em esforços de melhoria de qualidade, são economizados 10 dólares em custos. Portanto, as empresas de manutenção de software devem ter um foco contínuo em melhoria das métricas de serviço de suporte e redução de custos para melhorar a satisfação do cliente.

O quarto problema a ser considerado, é a demanda crescente por projetos de *Outsourcing*[1] nas empresas, no qual a principal fonte de renda destas empresas não é o de

[1] Ação de uma organização para obter mão-de-obra de fora da mesma, ou seja, mão-de-obra terceirizada.

tecnologia da informação, apesar de ter um papel fundamental para as suas sobrevivências no cenário econômico. Dentre inúmeros projetos de *Outsourcing* destacam-se os projetos de Gerenciamento de Serviços de Tecnologia da Informação (TI) ou *Information Technology Service Management* (ITSM) que estão relacionados a uma demanda crescente no uso desses serviços, citando os serviços de computação de operadores de telefonia, serviços de provedores de conteúdo, serviços bancários, serviços de computação em nuvem, apenas para citar alguns (ASSUNÇÃO et al. 2012). Essa necessidade dos clientes promove uma demanda de profissionais. Uma agilidade no processo de suporte com a utilização de ferramentas de automação, classificação e resolução de falhas, além de um processo otimizado de mudanças, pode suprir essa necessidade, ao invés da contratação pura e simples de mais profissionais para suprir a demanda.

Resumindo, quatro problemas principais de manutenção foram relatados: Qualidade, Complexidade, Custo e Demanda. Esses problemas são importantes e não triviais para equacioná-los porque refletem tanto a sobrevivência das empresas ou pessoas físicas que dependem destes serviços de manutenção e hospedagem de software, como as empresas que disponibilizam estes serviços.

Neste contexto, um dos grandes desafios das empresas que disponibilizam os serviços de manutenção e hospedagem de software é o de melhorar os Indicadores Chave de Desempenho (*Key Performance Indicators* - KPIs), entre os quais pode ser citado o Indicador de Tempo Médio para Resolver incidentes de serviço (*Mean Time to Resolve* - MTTR), que reflete a qualidade, verificando se o Acordo de Nível de Serviço (*Service Level Agreement* - SLA) junto ao cliente está em nível satisfatório e se os Objetivos de Nível de Serviço (*Service Level Objectives* - SLO) junto ao provedor estão também em nivel satisfatório.

Com relação à complexidade dos incidentes, busca-se cada vez mais um mecanismo de prevenção de defeitos de modo que possa haver uma redução no índice de incidentes e um aumento da disponibilidade na etapa de manutenção do sistema.

O custo e a demanda crescente pelos serviços de manutenção e desenvolvimento de software podem ser diminuídos com uma melhoria no processo de atendimento (qualidade), que é consequência de um tempo menor na resolução de incidentes e prevenção de defeitos.

Com relação ao estado da arte da classificação de defeitos, reconhecendo a importância desse estudo, muitas caracterizações de falhas foram realizadas, mas muitos desses estudos datam dos anos 80 e do início dos anos 90 (SAHOO et al., 2004).

Recentemente, vários artigos científicos se focaram em melhoria da qualidade na manutenção de software. Existem abordagens que incluem engenharia reversa de código no processo de desenvolvimento para minimizar *bugs* usando redes neurais (PARKA; OHB; PEDRYCZ, 2011). Outras abordagens definiram uma mineração de dados (*Data Mining*) de *bugs*. Nessas abordagens de prevenção de defeitos, um completo processo de mineração de dados é usado a partir de uma base de dados organizada em um *Data Warehouse*. (SCHUGERL; RILLING; CHARLAND, 2008). Em Oyetoyan, Cruzes e Conradi (2013) foi usado um método para classificar componentes de software extendendo métricas de orientação a objetos, com dependências cíclicas e não cíclicas. As dependências ciclicas seriam indícios de componentes de softwares propensos a erros. Outros estudos se focaram em modelos de classificação, como a Classificação Ortogonal de Defeitos, ou *Orthogonal Defect Classification* (ODC), para melhoria de processos. Nesta área destaca-se um processo de classificação automática utilizando algoritmos de mineração de dados em texto a partir de registros de incidentes (HUANG et al. 2011) e outro que utiliza algoritmos genéticos para associar recursos técnicos e soluções aos incidentes (XIAO e AFZAL, 2010). A pesquisa de Thung, Lo e Jiang (2012), propôs uma classificação de defeitos utilizando-se o ODC em três grupos: controle e fluxo de dados, defeitos estruturais e defeitos não funcionais de acordo com o relatório de *bugs* e com as mudanças de código para resolver um defeito. Em Cotroneo, Pietrantuono e Russo (2013), o ODC foi utilizado para melhorar técnicas de teste a partir do atributo ODC (tipos de falha) e para melhorar as métricas de software a fim de maximizar a identificação do número de falhas antes das fase de implantação e de produção. Relações entre o ODC e os tipos de *bugs* foram desenvolvidas por Chillarege (2011; 2013). Estudos, relacionados em outras áreas, usando o ODC foi encontrados na indústria têxtil por meio de um sistema de gerenciamento de qualidade usando processamento analítico online e regras de associação (LEE et al., 2013).

Embora vários trabalhos de pesquisa tenham se direcionado para as fases de desenvolvimento de software anteriores à fase de produção (produto entregue e em funcionamento) em provedores de serviço, de fato, custo e qualidade são pontos chave para a garantia de sucesso em provedores de serviço que se baseiam em fatores como qualidade no atendimento, implantação eficiente de aplicações, alta disponibilidade, bom suporte ao diagnóstico do sistema e processos que controlem a qualidade de recursos e custos.

A importância do tema em relação aos problemas apresentados no início desta seção, o estado da arte atual em relação à classificação de problemas e, por fim, a possibilidade de generalizar uma solução para outras empresas, motivaram a realização desta pesquisa. A busca da melhoria da qualidade e do atendimento, da diminuição da complexidade e do custo e o aumento da demanda, problemas apresentados anteriormente, justificam este trabalho. Verificar se uma classificação de defeitos extendendo o ODC é necessária ou suficiente ou os dois para diminuir ou não o tempo de atendimento dos incidentes fez parte dessa busca. Também fez parte dessa busca, verificar se um ranqueamento de soluções é necessário ou suficiente ou os dois para diminuir ou não o tempo de atendimento dos incidentes.

Baseado na justificativa, este deste projeto verificou se um método de resolução de incidentes e problemas, que utilize uma ranqueamento de defeitos/problemas, uma correta distribuição de incidentes para os times corretos para resolvê-los e que associe um conjunto de soluções para as classes de defeitos/problemas, pode reduzir:

- O tempo de resolução de incidentes (Qualidade e Complexidade).
- O custo de tempo para a restauração do serviço e os recursos humanos envolvidos para esta tarefa, provendo ainda uma solução que não aumente o custo total do provedor de serviço (Custo e Demanda).

1.1 *Objetivos*

1.1.1 Objetivo geral

Implementar e testar artefatos para a coleta dos dados de soluções de defeitos, utilizando-se um modelo de dados e um mecanismo de ranqueamento de soluções a partir dos dados classificados com a ajuda do modelo ODC, e de novas extensões do mesmo, que visem demonstrar que o método proposto é capaz de minimizar o problema do acordo de nível de serviço, diminuindo o tempo de resolução de incidentes.

1.1.2 Objetivos específicos

Com o desenvolvimento do trabalho espera-se:

- Verificar se a classificação correta das ordens de serviço utilizando o ODC, com novas extensões, é suficiente para separar de forma clara os tipos de incidentes ou problemas.

- Validar se os problemas mais fáceis de identificar (*bugs* determinísticos) podem ser resolvidos pelo primeiro nível técnico de suporte.
- Verificar se a base de conhecimento utilizada nas soluções associadas à classe de incidentes é suficiente para diminuir o tempo de atendimento a problemas mais complexos (*bugs* não determinísticos) no nível mais especializado de suporte.
- Validar se o método de ranqueamento de soluções classificadas pelo ODC pode diminuir o tempo de resolução de incidentes.

1.2 Estrutura do Trabalho

A presente dissertação está estruturada entre os Capítulos 2 ao 6, além dos apêndices. No Capítulo 2, é apresentado uma revisão bibliográfica a respeito do conhecimento divulgado sobre o tema, quais as estruturas e atributos que interessam e quais as que não interessam para a pesquisa, além do método de classificação ODC. No Capítulo 3, está explicado como foi desenvolvido o trabalho, incluindo os métodos utilizados para a revisão sistemática e para a fundamentação teórica da pesquisa. No Capítulo 4 são apresentados os resultados principais da revisão sistemática e os resultados do método proposto. O Capítulo 5 apresenta a discussão dos resultados obtidos. No Capítulo 6, há uma conclusão sobre a revisão sistemática, sobre o método proposto e indicação de futuros trabalhos. No Apêndice A - Revisão Sistemática, é explicada toda a revisão sistemática em detalhes e os resultados das chaves de pesquisa que foram a base para a fundamentação teórica do Capítulo 2. Por fim, no Apêndide B é mostrado um artigo publicado referente a dissertação.

2 REVISÃO BIBLIOGRÁFICA

Sobre o conhecimento divulgado a respeito do tema, foi realizada uma revisão bibliográfica na qual foram observados vários enfoques, sendo que alguns foram usados e outros foram descartados no presente trabalho.

Com relação à classificação de defeitos, reconhecendo a importância desse estudo, muitas caracterizações de falhas foram realizadas, mas muitos desses estudos datam dos anos 80 e do início dos anos 90, época em que software e hardware eram significantemente diferentes em relação ao design (SAHOO et al., 2004). Um dos materiais iniciais na classificação de defeitos está em Knuth (1989), no qual a partir de experiências em projetos de médio porte, tais como um sistema de composição tipográfica, foi utilizada uma categorização que associava classificação de falhas com melhoramentos. Estudos descritos em Beizer e Vinter (2001) provem um esquema de categorização de falhas usando um modelo hierárquico com dez categorias iniciais e cem categorias finais para o processo de desenvolvimento de software. Outros métodos de classificação de defeitos tais como em Eisenstadt mostravam ser muito ambíguos e o método de DeMillo era muito focado em código procedural (PLOSKI et al. , 2007). Embora tais métodos tenham contribuído para a melhoria da categorização de software, eles apresentavam muitas dificuldades de implantação na prática, devido ao alto nível de ambiguidade em várias falhas de software.

Estudos em Gray (1986) introduziram uma hipótese de *bugs*, chamados *bugs* determinísticos (*Bohrbugs*) e *bugs* não determinísticos (*MandelBugs*) conforme será explicado na seção 2.3.3 (Bug determinístico e não determinístico).

Ainda no trabalho citado em Sahoo et al. (2004), identificou-se que falhas não são uniformemente distribuídas e uma pequena fração dos servidores (4%) possuíam mais de 70% das falhas. Essas falhas, segundo o estudo, tinham uma relação direta com a hora do dia e variavam de acordo com o tempo e diferentemente entre os servidores.

O ITIL (*Information Technology Service Library*) surgiu posteriormente para descrever os processos de Gerenciamento de Incidentes (*Incident Management*), Gerenciamento de Mudanças (*Change Management*), Gerenciamento de Configurações (*Configuration Management*) e Gerenciamento de Versões (*Release Management*), normatizando os Serviços de Manutenção de Software (*ITSM - Information Technology*

Service Management), sendo o *framework* mais usado atualmente. (SAHOO et al., 2004). Porém o ITIL não lida com classificação de defeitos (ITIL OFFICIAL SITE, 2013).

Para preencher essa lacuna, entre a prevenção de defeitos e os processos de ITIL, muitas pesquisas foram e estão sendo direcionadas para a prevenção de defeitos com diferentes perspectivas. Existem pesquisas como a Análise de Falhas Baseadas em *Logs* ou *Field Failure Data Analysis* (FFDA) (PECHIA et al., 2011). Outras pesquisas foram encontradas em Ejarqueet al. (2008), usando sistemas multiagentes para soluções de incidentes de TI, onde a inteligência artificial é usada para gerenciar totalmente um sistema de suporte a incidentes baseado nos princípios de ITIL.

Existem abordagens que incluem Engenharia Reversa de Código no processo de desenvolvimento a fim de minimizar *bugs*, utilizando diversos algoritmos como regressão liear e redes neurais (PARKA; OHB; PEDRYCZ, 2011). Essas abordagens não cobrem o processo de manutenção de software.

Outras abordagens incluem a Mineração de Dados (*Data Mining*) de *bugs*. Nestas abordagens de prevenção de defeitos, um processo completo de mineração de dados é usado a partir de uma base de dados organizada em um *Data Warehouse.* (SCHUGERL; RILLING; CHARLAND, 2008).

Ainda sobre a classificação de defeitos, foi observada, em vários trabalhos, uma verificação da importância dessa classificação, de forma a facilitar a priorização dos defeitos importantes para o negócio, como, por exemplo, os defeitos que, corrigidos, permitem a melhoria do processo nos provedores de serviço e a diagnose das causas (PODGURSKI et al., 2003).

Há uma dificuldade em se classificar os defeitos em relação ao número e tipo de defeitos. Isto implica que a maioria das definições usadas faz com que não haja garantia de que dois diferentes indivíduos, olhando para o mesmo conjunto de falhas e a para as mesmas definições de falhas, farão uma contagem de mesmo valor. A esse respeito, encontram-se trabalhos que são focados em classificação baseada na versão do software (MUNSON; NIKORAB; SHERIF, 2006).

Outros métodos de classificação de software, para uma melhor detecção de erros, se baseiam no código fonte (código estático) como em Gondra (2008), que provê uma evidência empírica que correlaciona algumas métricas e erros de software. Por usar essas métricas, o prognóstico de defeitos de software é usualmente visto como uma classificação binária, na

qual os módulos de software são classificados em Inclinados a Falhas (*Fault-Prone*) e Não Inclinado a Falhas (*Non-Fault-Prone*).

Os métodos mais usados para a classificação de defeitos são o IEEE 829 (IEEE 829, 2013) e o *Orthogonal Defect Classification*, ou ODC (IBM RESEARCH, 2013). O IEEE 829 especifica a forma de uso de um conjunto de documentos em oito estágios de teste de software, cada estágio potencialmente produzindo o seu próprio tipo de documento. O ODC foi um conceito formulado no centro de pesquisas IBM no início da década de 1990. O foco inicial era ajudar o processo de retorno (*feedback*) dos dados atuais de defeitos para desenvolvedores e testadores. O esquema original foi primeiramente elaborado na cidade de *Poughkeepsie/NY* (EUA), a fim de refinar o esquema de classificação, bem como validar a análise de resultados (IBM RESEARCH, 2013). O ODC foi escolhido pela sua maior quantidade de artigos publicados, de acordo com a revisão sistemática, que pode ser verificada no Apêndice A - Revisão Sistemática. O modelo de tipo de *bug* foi selecionado como uma parte adicional ao método ODC por permitir uma distribuição dos incidentes entre os times de forma mais coerente, conforme está explicado na seção 2.3.3 (*Bug* determinístico e não determinístico).

2.1 *ODC*

A Classificação Ortogonal de Defeitos, ou *Orthogonal Defect Classification* (ODC), inicia-se com uma proposta de classificação de defeitos para a melhoria do processo no ciclo de desenvolvimento de software, como apresentado em Chillarige et al. (1992). Essa pesquisa, que abordou pela primeira vez o termo ODC, propôs, como idéia central, extrair informações semânticas a partir de defeitos. A partir dessas informações foi extraída uma relação de causa e efeito no processo de desenvolvimento de software. Ainda em Chillarige et al. (1992), com relação à causa dos defeitos, foram classificados os atributos que descrevem os defeitos. No defeito, são medidos os atributos ou medidas que tem alvo no produto ou processo além da severidade ou impacto junto ao cliente.

Alem disso, em Chillarige et al. (1992) é apresentado todo o modelo ODC, destacando-se o tipo de defeito e o gatilho do defeito, propondo que a extensão natural do ODC é o Processo de Prevenção de Defeitos ou *Defect Prevention Process (DPP)*. A experiência com o ODC, segundo o artigo, provê um retorno rápido aos desenvolvedores e o

gatilho do defeito prove uma medida de eficácia para o estágio de verificação. Isso foi obtido através de experimentos que foram apresentados no referido artigo.

A partir desse artigo inicial, muitas outras publicações surgiram como em IBM Research (2013). Nesse centro de informações, o ODC é definido como uma tecnologia rica e multifacetada, que prove muitos benefícios. Alguns benefícios que os times de suporte comumente usam são:

- **Prevenção de defeitos.** Somente por diminuir o número de defeitos que existem no projeto e no código, pode-se diminuir o tempo e os custos de desenvolvimento. Como foi demonstrado em pesquisas em mais de 60 projetos dentro da IBM e em outras companias como Motorola e Nortel, o ODC ajuda a identificar onde os defeitos são introduzidos, de forma que os times podem identificar ações para melhorar os requisitos, o design e as atividades de codificação, fazendo com que poucos defeitos sejam gerados durante essas atividades (IBM Research, 2013).
- **Remoção de defeitos.** O ODC ajuda a melhorar a eficácia e a eficiência dos testes, por identificar quais as condições que conduzem a fugas de defeitos para produção. Com isso, testam-se essas fugas antes de se levar o software a um ambiente de produção.

O ODC usa uma série de atributos ortogonais para classificar um defeito (CHILLAREGE et Al., 1992), que incluem a atividade, o gatilho, o impacto, o alvo, o tipo de defeito, o qualificador de defeito, a fonte e a idade

Hoje, o ODC tem sido largamente usado em muitas companhias de software incluindo IBM, Motorola, Nortel e Lucent (HE; HAO; ZHIQING, 2009). Muitas pesquisas do ODC estão sendo focadas na aplicação de gerenciamento de defeitos. Algumas delas, como em Saraiya et Al. (2006), fazem uma matriz harmônica para melhorar o processo de coleção de dados. Consultando a opinião de especialistas no assunto, os autores construíram essa matriz harmônica e mostrou as relações entre os vários tipos de gatilhos e defeitos, identificando três categorias de relações (baixa, média e alta) existentes entre os mesmos. Segundo He, Hap e Zhiqing (2009), a fraqueza dessa técnica está no fato de a mesma considerar apenas dois atributos, o gatilho e o tipo de defeito, sendo que, na verdade, outros atributos também têm influência uns sobre os outros, e a técnica não faz uso de históricos de dados de forma eficiente para se sustentar.

Ainda segundo He, Hap e Zhiqing (2009), o melhor método seria o de utilizar uma rede bayesiana para investigar a relação entre os atributos de causa e efeito do ODC. Esse modelo utiliza dados históricos de defeitos, de uma empresa de porte médio, relativos aos desenvolvedores e ao time responsável pelo teste de software.

Na pesquisa de Thung, Lo e Jiang (2012), foi proposta uma classificação de defeitos dividindo o ODC em 3 famílias: controle e fluxo de dados, defeitos estruturais e defeitos não funcionais de acordo com o relatório de *bugs*, ou seja, indicando a explicação do defeito encontrado e as mudanças de código para resolver o defeito. Dado um grande conjunto de dados sobre defeitos, as soluções em formato de texto foram extraídas a partir da descrição dos *bug*s. Além disso, vários elementos foram extraídos a partir dos códigos que consertaram os *bug*s. Baseado em partes das informações coletadas, treinou-se um modelo discriminante que classifica o defeito dentro de uma das três categorias descritas acima para depois classificar o conjunto de testes em 500 defeitos coletados utilizando-se maquinas de suporte vetorial. Apesar da acurácia relatada de 77,8 %, a classificação dos defeitos é muito genérica, baseada totalmente em código fonte. Esta pesquisa muitas vezes não se encaixa em casos relacionados a requisitos não funcionais, como erros de produção relacionados à configuração de software, que são situações comuns em provedores de serviço.

Em outras pesquisas, como em Huang et al. (2011), há métodos de classificação de defeitos de software que descrevem um processo automático chamado AutoODC, utilizando um mecanismo de texto estruturado dos incidentes ocorridos e aplicando mineração de textos através do método de Maquinas de Suporte Vetorial ou *Supporting Vector Machines* (SVM). Um grande potencial de melhoria desse modelo está, principalmente, na área de pré-processamento.

Este modelo AutoODC (HUANG et al. 2011) e o bayesiano (HE; HAO; ZHIQING, 2009) são plausíveis, mas não são usados de uma forma isolada neste trabalho. Um método de classificação usando um modelo de *Data Warehouse* seria o mais adequado como descrito em Han, Kamber e Pei (2012). Isto porque os dados somente tem qualidade se os mesmos satisfazem os requisitos básicos de uso, ou seja, se os dados faltantes ou incorretos *(noises)* não são tratados, eles podem gerar erros para o classificador de mineração de dados, uma acurácia ruim, ou ainda uma mineração sem significado *(outliers)* . O *Data Warehouse* torna-se parte fundamental na preparação dos dados.

Existem muitos fatores que estão relacionados à qualidade dos dados, incluindo a acurácia, a completude, a consistência, a pontualidade, a credibilidade e a interpretabilidade. Por esses fatores fundamentais, para que se tenha qualidade nos dados, é que o pré-processamento se torna importante. Dentro dessa fase de pré-processamento, inclui-se a Limpeza dos Dados (*Data Cleaning*) para se evitarem dados inconsistentes, a Integração dos Dados (*Data Integration*) onde múltiplas fontes de dados podem ser combinadas, a Redução dos Dados (*Data reduction*) onde se obtém uma representação reduzida do conjunto de dados, muito menor em volume e que produz os mesmos resultados analíticos. Já na seleção de dados, os mesmos são selecionados por relevância a partir das bases de dados. Por fim, na fase de Transformação de Dados (*Data Transformation*), os mesmos são transformados e consolidados na forma apropriada. Somente a partir desse ponto, é que um processo de mineração de dados *(mining)* poderia ser adequado para a descoberta de conhecimento (HAN; KAMBER; PEI, 2012).

2.1.1 Fluxo de trabalho do ODC

Descreve-se o ODC como um modelo com quatro passos principais, que é chamado fluxo de trabalho (*workflow*) do ODC, como descrito em IBM Research (2013) e apresentado na Figura 2.

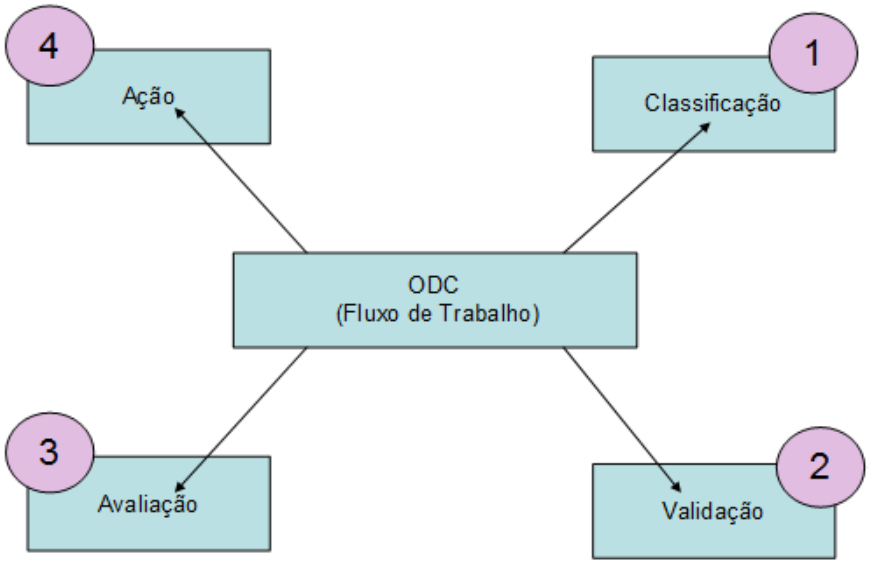

Figura 2 - Fluxo de trabalho do ODC, adaptado de (IBM RESEARCH, 2013).

Os passos, indicados na Figura 3, são os seguintes:

1 – Classificação (*Classify*): Na classificação, captura-se como o defeito foi descoberto, o efeito que teria ou teve para os clientes, o escopo e a escala na qual teve que ser consertado. Qualquer participante do processo, que abre ou fecha chamados para correção de defeitos, faz a classificação

2 – Validação (*Validate*): A validação, a qual é executada por um subconjunto do time de classificação, ajuda a assegurar que o passo de classificação foi feito de forma correta.

3 – Avaliação (*Assess*): A fase de avaliação dos dados serve para entender qual ação a ser tomada. É normalmente feito por um time bem pequeno no produto ou área na qual o defeito foi encontrado.

4 – Ação (*Act*): Nesta fase, há um envolvimento com atividades que requer muito trabalho. Nos três passos anteriores, somente se identifica o que precisa ser feito. Nesta fase é realmente executada a ação para corrigir o defeito. Identificar e programar essas ações requer conhecimento, determinação e gerenciamento de suporte, para que a mudança necessária seja segura e não ocasione outros defeitos.

Os objetos de estudo para este trabalho de pesquisa são os passos de Classificação e Ação do ODC.

2.2 *Classificação e atributos usados a partir do ODC*

Na classificação ODC, como descrito em IBM research (2013), são usadas duas fases do ciclo de vida dos defeitos, a fase de abertura e a fase de fechamento do defeito. Na fase de abertura, há a figura do **apresentador** (*submitter*), que é qualquer pessoa que **abre um defeito**. Dependendo da organização e do processo, pode ser um desenvolvedor, uma pessoa de testes, uma pessoa que presta um serviço ou mesmo um gerente. O importante é que essas pessoas saibam o que estavam fazendo quando o defeito foi exposto a partir de uma falha. O recurso que vai atender ao chamado de serviço (*responder*) é qualquer pessoa que **resolve o defeito**. Ela é, quase sempre, um desenvolvedor, um suporte de primeiro nível (*help desk*), um administrador de sistema ou um responsável por suporte avançado. O que importa é que uma pessoa sabe o que tem que ser modificado para corrigir o problema.

Nesta pesquisa são abordadas as duas fases do ciclo de vida dos defeitos, a abertura e o fechamento.

2.2.1 Abertura de um defeito

Em geral, quando um defeito na manutenção de software é aberto, um campo sobre a descrição do defeito é preenchido e uma severidade é relacionada ao mesmo.

Utilizando a classificação ODC na abertura de um defeito, os atributos de estudo são os seguintes, como descrito em IBM Research (2013):

- Fase Encontrada. Refere-se à fase escalonada de teste em que o usuário está quando o defeito é encontrado. Se um defeito de um sistema é achado depois que foram feitos os testes e o mesmo já está em produção, então a fase é usualmente chamada de campo, manutenção ou produção. Durante o desenvolvimento, a fase refere-se ao planejamento de projeto, calendário ou linha do tempo e descreve a fase na qual o gerente de projeto diz que o usuário está.

- Atividade. Diz respeito à atividade atual que foi executada quando um defeito é descoberto.

- Gatilho. Corresponde ao ambiente ou condição para encontrar o defeito. Os gatilhos são os catalisadores para permitir que os defeitos de superfície apareçam. Em resumo, descreve-se como o defeito foi encontrado. Capturar a informação de gatilho é provavelmente a parte mais difícil na classificação dos defeitos. Quando se classificam os defeitos de campo, inicia-se com o gatilho, selecionando o gatilho que melhor representa o ambiente ou a condição que expôs o defeito no ambiente do cliente. A atividade é então selecionada, com base no gatilho que foi mapeado.

- Impacto. Está relacionado ao tipo de influência que o cliente vai sofrer, caso o defeito não seja encontrado.

2.2.2 Fechamento de um defeito

Em geral, quando um defeito no gerenciamento de serviços de suporte de incidentes é fechado, um campo de “Resolução do Defeito” e outro de “Data de Fechamento” são preenchidos, muitas vezes, automaticamente. Nesta fase são obtidos todos os detalhes do defeito.

No caso da classificação ODC, quando ocorre o fechamento de um defeito, o desenvolvedor ou a pessoa que corrigiu o erro submeterá os cinco atributos descritos a seguir ao modelo ODC, como descrito em IBM Research (2013):

- Alvo (*Target*). Mostra o que foi corrigido, sendo uma declaração genérica. Podem ser alvos, por exemplo: Requisitos, Código, Projeto *(Design)*, Desenvolvimento de Informação, Pacote, Versão *(Build)*, Suporte à Linguagem, entre outros.

- Tipo de defeito. O tipo de defeito mostra qual o tipo de correção que será utilizada para consertar o defeito, não significando como corrigir o defeito, mas a natureza da correção.

- Qualificador. É uma definição adicional do tipo de defeito, indicando se o elemento do defeito era não existente, incorreto ou irrelevante.

- Fonte (*Source*). Indica a fonte do elemento corrigido, ou seja, se foi desenvolvido na própria empresa, reutilizado a partir de uma biblioteca, obtido de alguma outra companhia ou feito uma portabilidade de outros ambientes no caso de migração de sistema de uma plataforma a outra.

- Idade *(Age)*. A idade identifica o histórico de edição da entidade corrigida. Os tipos são:

- **Novo**. O defeito é de uma função nova que foi criada para o projeto em questão.
- **Base/versão anterior**. O defeito está em alguma parte do produto que não foi modificado para o projeto corrente, e não é parte de uma biblioteca padrão de reuso.
- **Reescrito**. O defeito foi introduzido como resultado de uma alteração no projeto ou reescrito a partir de uma função antiga, com o intuito de melhorar o design ou a qualidade.
- **Reconsertado *(Refixed)***. O defeito foi introduzido para solucionar um defeito anterior.

2.2.3 Bug determinístico e não determinístico

Um conceito importante na fundamentação teórica do processo proposto neste trabalho, mas que são esta contido no ODC, é a verificação se o *bug* é determinístico (*"Bohrbug"*) ou não determinístico ("*Mandelbug"*). Esta importância se deve ao fato que, com esta classificação, pode-se direcionar o incidente de suporte para diferentes times. Pode-se encaminhar os *bugs* determinísticos para times de suporte com menos conhecimento e os *bugs* não determinísticos para os times de suporte mais qualificados. Os conceitos, em mais profundidade, são explicados a seguir.

O termo *"Bohrbug"* foi primeiramente usado em Gray (1986) e está relacionado a defeitos de software sólidos, isto é, defeitos que são facilmente detectados e corrigidos, onde as ocorrências de falha são facilmente reproduzidas. O conceito reproduzido em Grottke e Trivedi (2005) define que ***Bohrbug*** é um defeito que é facilmente isolado, e que se manifesta consistentemente sobre um conjunto de condições bem definido, porque sua ativação e propagação possuem uma baixa complexidade. Ainda em Grottke e Trivedi (2005), ***Mandelbug*** é um defeito no qual a ativação ou a propagação do erro são consideradas complexas. O termo complexo pode tomar duas formas:

- A ativação e/ou propagação dependem das interações entre as condições que ocorrem dentro da aplicação e condições que se acumulam dentro do sistema no ambiente interno da aplicação.

- Existe uma defasagem de tempo entre a ativação do defeito e a ocorrência da falha, porque vários estados de erro diferentes são percorridos na propagação do erro.

Tipicamente, um *mandelbug* é difícil de isolar e/ou os defeitos são causados por uma causa que é difícil de ser reproduzida. Como descrito em Grottke e Trivedi (2005) fala-se no termo *Heisenbug*, o qual é um subconjunto de *mandelbug,* no qual um defeito para de gerar uma falha ou se manifesta diferentemente quando se tenta debugar ou isolar o erro. Outro subtipo de *mandelbug,* como encontrado em Grottke e Trivedi (2005), é o erro relacionado ao envelhecimento ou *Aging-related bug,* que é um defeito que leva a um acúmulo de erros, tanto dentro da aplicação como em um ambiente interno do sistema, resultando em um aumento da razão de falha e/ou em uma degradação de desempenho.

Na Figura 3, como descrito em Grottke e Trivedi (2005), é apresentado um diagrama de Venn sobre esses quatro tipos de *bug*.

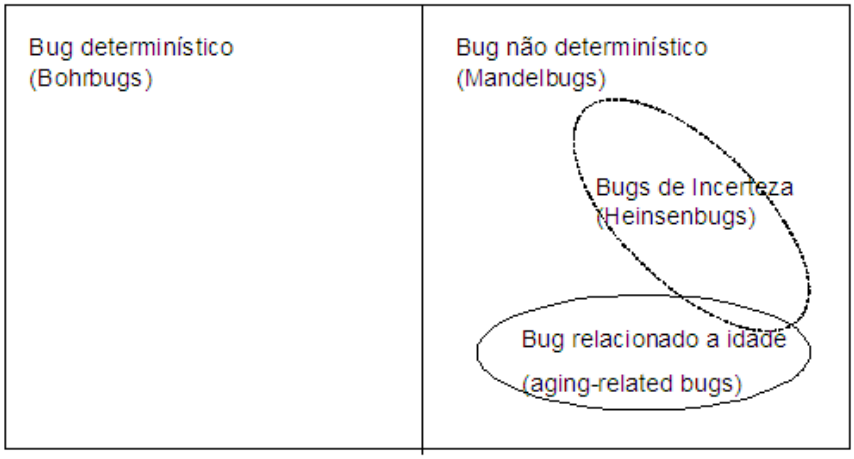

Figura 3 - Diagrama de Venn para os tipos de bug, adaptado de (GROTTKE; TRIVEDI, 2005)

Em Chillarege (2011), tem-se uma divisão de *bohrbugs* e *mandelbugs* por grupos de gatilhos do ODC, de acordo com o Quadro 1.

Gatilhos de bugs determinísticos (Bohrbugs)	**Gatilhos de bugs não determinísticos (Mandelbugs)**
Compatibilidade Lateral	Concorrência
Compatibilidade com versões anteriores	Situações raras
Cobertura	Efeitos colaterais
Variação	Iteração
Conformidade de Design	Seqüência
Fluxo Lógico	Exceção/Recuperação
Modo Normal	Carga de Trabalho *(Workload)/Stress*
Caminho Complexo	
Configuração de *Hardware*	
Configuração de *Software*	

Quadro 1 - Tipos de gatilho agrupados em bohrbugs e mandelbugs. (CHILLAREGE, 2011)

A partir desta relação de tipos de *bugs* e gatilhos, um estudo de caso foi feito com softwares de missões espaciais, revelando que os *bugs* não determinísticos (*Mandelbugs*) aconteceram num total de 20% a 40% do total de *bugs* (CHILLAREGE, 2011). Em outro estudo complementar de Chillarege (2013), incluiram-se mais duas dimensões, o impacto ODC e a fase encontrada ODC do problema nos quatro casos de estudo verificados na

pesquisa anterior. Os resultados mostraram que os *bugs* não determinísticos (*Mandelbugs*) mostram um impacto predominantemente em confiabilidade, performance e manutenção para o cliente, raramente gerando um impacto funcional para o cliente. *Mandelbugs* apareceram em 30% a 40% dos *bugs* em produção e 10% dos *bugs* nos testes de qualidade.

2.3 *Métodos de mapeamento de melhores soluções para uma classe de problemas*

As melhores soluções, e o seu ranqueamento para uma classe de problemas, são parte importante desta pesquisa. Em Podgurski et al. (2003) mostrou-se um mecanismo onde uma classificação de defeitos se propõe a priorizar defeitos importantes para o negócio, de forma a melhorar o processo do provedor de serviços e o diagnóstico de causas raiz. Mostra-se que são causadas por um mesmo defeito e um considerável número de ocorrências que podem ser elegíveis para a prevenção desses tipos de defeitos. De acordo com a criticidade das falhas, deve-se priorizar a investigação dos incidentes após a restauração (Controle de Problemas) de forma que se evite os mesmos defeitos novamente. Também é usado um cluster manual para definir a similaridade de falhas em um espaço bi-dimensional, baseado na distancia entre os pontos a partir de determinados critérios. Grupos de falhas podem ser identificados a partir de uma inspeção visual. A desvantagem desse modelo é que somente duas dimensões são expostas e poucas diferenças entre as falhas são apresentadas. O ODC poderia ser usado nesse modelo porque é uma classificação multidimensional de defeitos.

Na pesquisa de Wang, Akella e Ramachandran (2010) é proposto um sistema de análise de texto chamado Analisador e Recomendador de Requisições de Serviço ou *Service Request Analyzer and Recommended* (SRAR), que extrai informação crítica e pertinente a partir de documentos em texto de requisições de serviço de bases de dados de uma empresa de software usando técnicas de mineração, mas com uma inteligência de dados. Essa inteligência de dados é formada por um conjunto de palavras (*Bag of words*), um domínio de conhecimento e um fator de iteração de especialistas. Descrevendo sucintamente o processo, a partir de um pré-processamento, um classificador hierárquico de mineração de dados e o algoritmo Naive Bayes são aplicados para categorizar o capital intelectual e remover o conteúdo irrelevante. Um método de gerador de características incorpora o domínio de conhecimento, expandindo as terminologias e melhorando as características de extração de texto. Um método de aprendizado foi incorporado ao processo para acomodar situações nas

quais os recursos envolvidos na solução do problema não estão de acordo com a extração de capital intelectual, e mostra-se outras requisições de serviço com conteúdo similar. Uma pesquisa anterior a esse método pode ser encontrada no Raciocínio Baseado em Casos ou *Case-Based Reasoning,* que é um paradigma de resolução de problemas a partir de casos similares, reutilizado informações para resolver um determinado problema (Aamodt and E. Plaza, 1994).

2.4 *DataWarehousing e Data Warehouse*

Data Warehousing é o processo de recuperação e integração de dados a partir de fontes de dados distribuídas, autônomas e, muitas vezes, heterogêneas. Segundo Inmon e Kelley (1992), um Data Warehouse é uma coleção de dados orientada por assuntos, integrada, variante no tempo e não volátil, que tem por objetivo dar suporte aos processos de tomada de decisão. Uma tradução livre para *Data Warehouse* seria Armazém de Dados ou Depósito de Dados. O termo Data Warehouse foi utilizado pela primeira vez por William Harvey Inmon, em 1992 (INMON; KELLEY, 1992).

O objetivo principal do processo de *Data Warehousing* é possibilitar a transformação de uma base de dados de uma organização, geralmente feita para Processamento de Transações em Tempo-real (*Online Transaction Processing* - OLTP) em para uma base de dados que contenha o histórico dos dados existentes na organização para Processamento Analítico em Tempo-real (*On-line Analytical Processing* - OLAP), também conhecido como *Data Warehouse*.

O *On-line Analytical Processing* (OLAP) pode analisar um grande volume de dados sob várias perspectivas. As aplicações OLAP são utilizadas para suporte à tomada de decisões por gestores.

Segundo Kimball (1998), os objetivos de um Data Warehouse são:

- O Data Warehouse deve fornecer acesso aos dados organizacionais.
- Os dados no Data Warehouse devem ser consistentes.
- O Data Warehouse é um local onde são publicados dados confiáveis.
- A qualidade dos dados no Data Warehouse depende dos dados utilizados para criar o mesmo, ou seja, o Data Warehouse não aprimora dados de baixa qualidade.
- Os dados do Data Warehouse podem ser separados e combinados (slice e dice).

- O Data Warehouse não consiste apenas de dados, mas também de um conjunto de ferramentas para consultar, analisar e apresentar informações.

Para Inmon (2012), os requisitos básicos para um *Data Warehouse* são:

- Deve ser organizado por assuntos, ou seja, armazenar informações sobre temas específicos importantes para o negócio.
- Deve possuir capacidade de integração, ou seja, garantir a consistência e uniformidade para as informações por meio da padronização das mesmas.
- Não é volátil, ou seja, o Data Warehouse deve se referir a informações em algum momento específico, não sendo atualizável nas podendo ser incrementado.
- Deve ser flexível.
- Os dados devem existir em vários níveis de granularidade.

2.4.1 Operações básicas

Podem ser feitas diferentes operações para poder visualizar e analisar os dados em um *Data Warehouse*. As operações que podem ser realizadas são (INMON, 2012):

- *Slice e Dice*.
- *Drill Down*.
- *Roll Up* (Consolidação).
- Rotação.

As operações de *Slice* e de *Dice* envolvem a redução do número de dados, mas mantendo a mesma perspectiva de visualização. Elas restringem os dados a serem analisados a um subconjunto desses dados. Esse recurso é utilizado para criar visões dos dados por meio de sua reorganização, de forma que eles possam ser examinados sob diferentes aspectos. Slice significa fatiar o cubo. Dice significa cortar em pequenos cubos, fazendo um sub-cubos para análise.

O *Drill Down* é uma técnica pela qual pode-se navegar de dados mais gerais, ou consolidados, para dados mais detalhados ou desagrupados.

O *Roll Up* é o inverso do *Drill Down,* sendo uma técnica pela qual pode-se navegar de dados mais detalhados ou desagrupados para dados mais gerais, ou consolidados.

A Rotação reorienta a visão multidimensional dos dados, oferecendo diferentes perspectivas dos mesmos dados.

2.4.2 Extração, transformação e carga

A Extração, Transformação e Carga (ETL- Extract, Transform and Load) é o processo de captação de dados das diversas fontes existentes na organização para instanciar o Data Warehouse. (INMON, 2012):

- A extração consiste na obtenção desses dados das diversas fontes disponíveis, sejam elas bases de dados, arquivos de texto ou arquivos binários.
- Os dados obtidos são então transformados do ambiente operacional para o ambiente do *Data Warehouse*, onde os dados são integrados, refinados, transformados e sumarizados.
- Já no processo de carga, o volume de dados processado é instanciado no *Data Warehouse*, onde ficará disponível para análise.

3 MATERIAIS E MÉTODOS

3.1 *Processo atual de atendimento dos incidentes*

A Empresa “A” possui uma abrangência global existente em vários países. Um dos focos deste empresa é prover serviços de suporte com gerenciamento de data center dos clientes dos mais diversos setores (Fabris, Bancários, Saúde, etc.). Este gerenciamento de data center de acordo com o contrato pode ser desde a venda de máquinas como um gerenciamento completo de problemas e mudanças de software.

Dentro de diversos processos da Empresa “A”, observando a Figura 4, o processo atual de atendimento dos incidentes é definido a partir de um gatilho inicial. Esse gatilho pode ser um gatilho manual, representado através incidentes de suporte reportados pelos usuários como, por exemplo, uma funcionalidade que esteja falhando, a performance do sistema que não está de acordo com o funcionamento normal, se o sistema como um todo estiver fora de operação, além de muitos outros tipos de chamadas. Este processo se refere ao passo 1 na Figura 4.

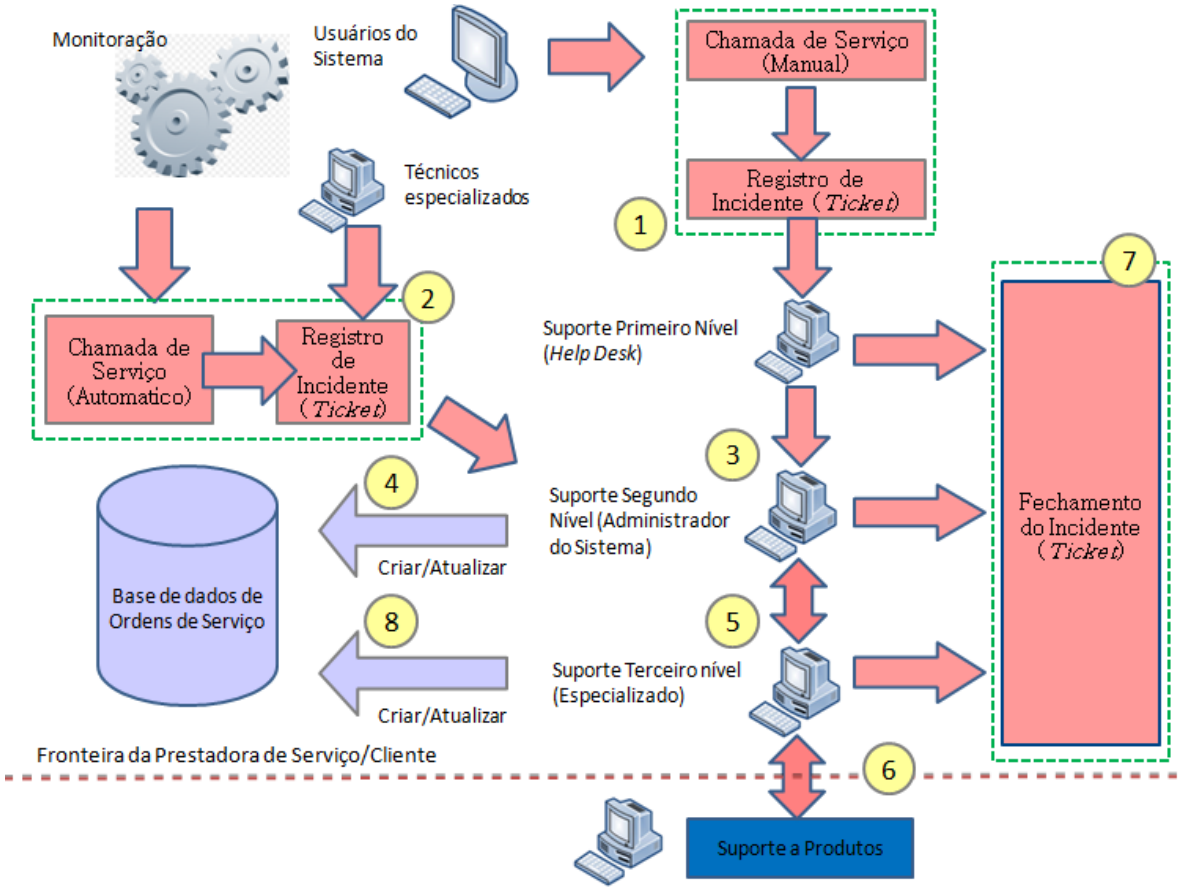

Figura 4 - Processo atual de atendimento de incidentes da Empresa A

Uma outra forma de disparo de gatilho (ver o passo 2 da Figura 4) seria um acionamento automático. Isso pode ser feito através de uma monitoração de uso excessivo de memória, de CPU, uma percentagem alcançada de uso em um sistema de arquivos, um limite do uso de paginação de memória, além de outras monitorações automáticas. Esse mesmo tipo de chamado pode ser aberto por um técnico que perceba algum problema (ver o passo 2 da Figura 4) de segurança ou algum desvio anormal no sistema. Tanto o passo 1 quanto o passo 2 criam chamados. O passo 1 passa pelo primeiro nível ou *help desk,* neste caso o atendente cria o chamado de acordo com a descrição do cliente. No passo 2 o ticket é gerado automaticamente ou criado sem a ajuda do *help desk* e vai para o suporte de segundo nível (ver o passo 3 da Figura 4).

Nos passos seguintes, o chamado de serviço pode ir para um nível mais especializado (ver o passo 5 da Figura 4) e voltar caso a iteração entre os times seja utilizada. Quanto mais o chamado faz essa iteração entre os níveis, mais aumenta o tempo de atendimento. Um ponto importante é que o direcionamento automático ou do nível inicial pode direcionar erroneamente o chamado para um nível que não tem condição de resolvê-lo ou, ainda, um nível especializado é acionado sem necessidade, quando um nível menos especializado poderia resolver o mesmo chamado. Isso gera mais custos operacionais associados, como a hora gasta por um nível especializado ou o deslocamento do recurso humano para o local de atendimento.

A partir do último nível de atendimento da empresa, chega-se ao nível de atendimento do produto envolvido (ver o passo 6 da Figura 4). Neste caso, o limite da empresa é ultrapassado, podendo haver um atendimento dedicado pela empresa que faz o produto ou ainda um atendimento através da Internet como, por exemplo, alguma atualização de software disponível e que que possa resolver o chamado. Os passos 4 e 8 (ver a Figura 4) se referem à atualização do trabalho realizado nos registros de incidentes em uma base de dados de controle do provedor, que é a base de dados de ordens de serviço. O passo 7 (ver Figura 4) se refere ao fechamento do incidente, e pode ser realizado em qualquer fase do processo, a partir de um falso alarme ou quando o *Help Desk* (ver Figura 4) consegue resolver o chamado, ou ainda, mesmo que o chamado tenha alcançado o último nível de atendimento. Em um padrão aceitável de qualidade, o fechamento do incidente somente é feito com o aceite do cliente em relação a solução, ou quando o tempo de resposta de aceite chega a um limite de espera por

parte do provedor de serviços. As interações dos clientes ou recursos técnicos junto ao chamdo são representada por linhas tracejadas em verde (ver Figura 4).

3.2 *Problemas no processo atual dos incidentes*

Problemas foram encontrados no processo atual dos incidentes principalmente no que diz respeito a transferência entre os times em relação a complexidade e as informações disponíveis para resolução dos incidentes e problemas. Como regra geral os chamados sempre passam do nível menos especializado ao nível mais especializado.

Entretanto, a complexidade dos incidentes foi um ponto importante em uma análise diária. Isto foi obtido a partir de um trabalho dentro do próprio processo por vários anos e a troca de informações junto a companheiros de equipe. Há muitos casos em que o segundo nível de suporte (Número 3 da Figura 4) não tem condições de atender a um tipo de problema e gasta-se muito tempo para tentar resolvê-lo até chamar um nível especializado. Em outros casos o problema seria adequado ao segundo nível de suporte, entretanto, devido a fraca ou inexistente documentação a respeito de uma solução do problema torna-se complexo o bastante, sendo isto, um fator de transferência ao nível mais especializado. Estes problemas do segundo nível são muitas vezes repetitivos em que procedimentos específicos poderiam ser adicionados a um banco centralizado de soluções. Este fato gera problemas ao processo em que níveis mais especializados ficam sobrecarregados com problemas mais simples gerando carga de trabalho desnecessária e impedindo que os esforços dos níveis mais especialziados sejam direcionados a incidentes mais complexos e geranciamento de problemas, como aplicação de correções definitivas aos incidentes, correções de segurança, somente para citar alguns problemas.

Dentro do nível mais especializado (Número 5 da Figura 4), tem –se muitas vezes diferenças de conhecimento entre membros do time. Por exemplo, há situações em que um problema que foi resolvido por um recurso humano deste nível de suporte não é documentado ou repassado a membros do time. Quando chega um problema parecido ou idêntico a busca pela solução começa praticamento do início com o conhecimento pessoal que o recurso humano tem e não por uma base centralizada e classificada de informações.

Uma dificuldade que o modelo apresenta com relação a base centralizada de informações se resume a uma grande iteração entre componentes de serviços nos sisetmas das aplicações como firewalls, hardware, sistema operacionais com inúmeras versões, versões em

grande diversidade de bancos de dados, middlewares, linguagens de programação de aplicações as mais diversas e além disto arquiteturas de aplicações bem diferentes e com procedimentos bem distintos de configuração e manutenção.

A base centralizada das ordens de serviço possui dados em duplicidade e ambiguidades muito acentuada dificultando uma utilização da forma que foi concebida para poder dividir os problemas. Além disto dados vazios e muito incompletos dificultam uma análise histórica de problemas anteriores. Estes dados muitas vezes são mal documentados devido muitas vezes a uma baixa conscientização dos recursos envolvidos e a alta carga de trabalho durante o turnos de atendimento.

Os problemas de atendimento foram agravados devido a redução no quadro de funcionários para reduzir custos e devido a vários acordos de nível de serviço serem quebrados junto ao cliente e uma base de dados de soluções e decisões de transferências poderia melhorar o cenário atual de atendimento.

3.3 *Etapas da Pesquisa*

Foram realizadas as seguintes etapas da pesquisa de acordo com a Figura 5.

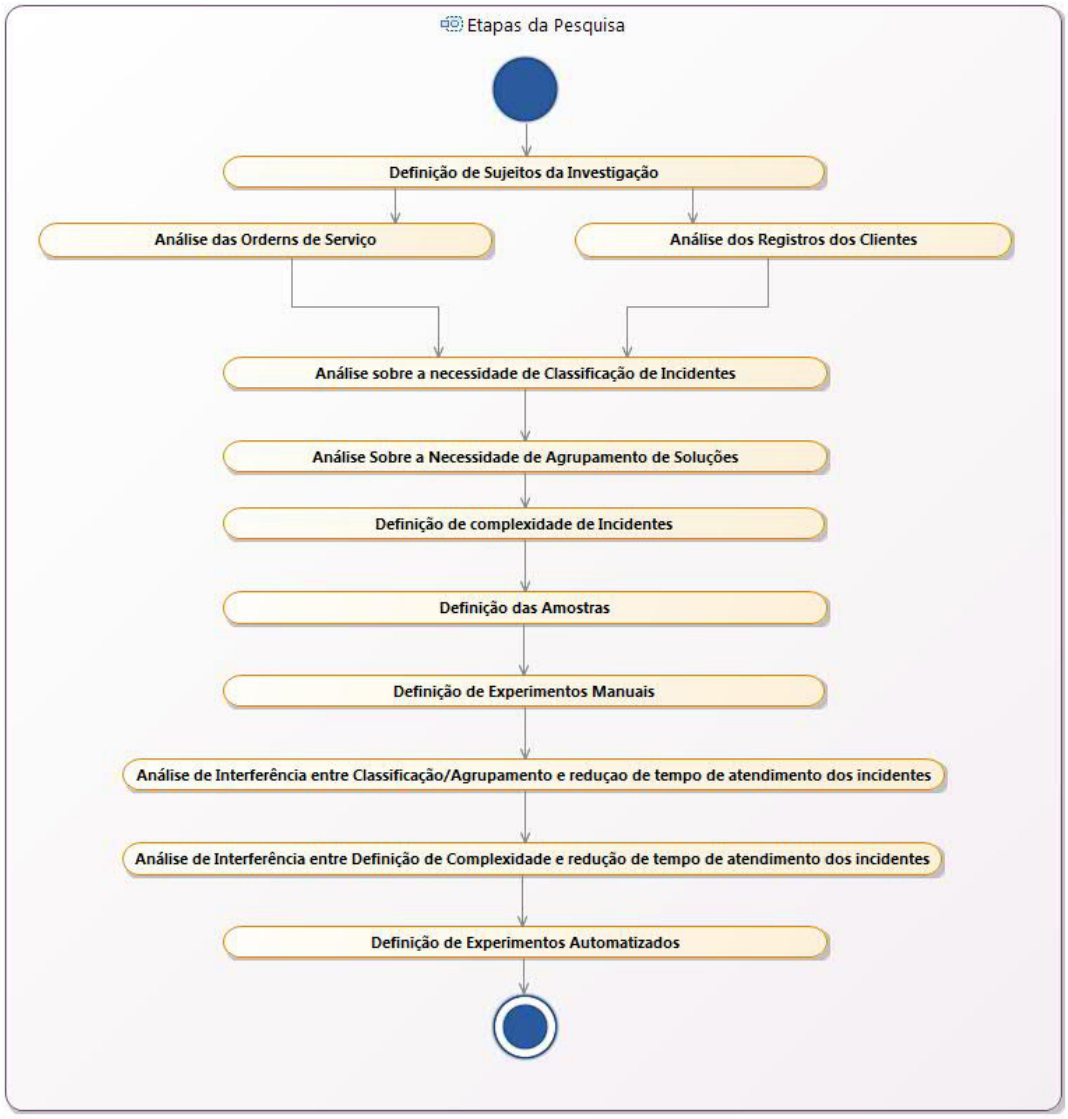

Figura 5 - Etapas da pesquisa

Estas etapas iniciam-se com o sujeito da investigação até a fase de análise de interferência entre as variáveis de classificação e de agrupamento de soluções e a variável de tempo (redução do tempo no atendimento dos incidentes). A seguir descreve-se cada etapa da pesquisa em detalhes.

3.4 *Definição de Sujeitos da investigação*

Na Figura 5, O sujeito da investigação consiste nas ordens de serviço *(workorders)*. Estas ordens de serviço são os registros do processo atual de atendimento dos incidentes. As

ordens de serviço são tipos de documentos que descrevem os incidentes de manutenção de software e que são informações inseridas a partir recursos humanos que atuaram no incidente, sendo que a sua condição prévia para este subconjunto de dados é que tenham alguma anomalia em relação ao comportamento normal do sistema. Este subconjunto é necessário porque existem atividades lançadas na ferramenta de ordem de serviço que não são atividades relacionadas a incidentes.

Adicionalmente, cada cliente tem uma ferramenta diferente de suporte que interage com o provedor de serviços. Essas ferramentas são sistemas internos de controle. Algumas ferramentas são conhecidas como Service Now e Manage Now e geram bases de dados de registros do cliente. Um outro sujeito da investigação portanto é o registro do cliente que são provenientes dessas bases de dados. Nesses registros são encontrados informações adicionais, como informações mais elaboradas das comunicações entre os times de suporte e das soluções para o incidente que foram empregadas.

Estes sujeitos são importantes a pesquisa no sentido que eles são as variáveis operacionais em que será aplicado o processo proposto para a diminuicão do tempo de atendimento.

3.5 *Análise das ordens de serviço*

Seguindo a definição de Sujeitos de Investigação (Figura 5), as ordens de serviço da Empresa “A” foram avaliadas com relação a consistência e completude de dados para o propósito de serem suficientes para atenderem ao objetivo de formar uma base de dados de problemas classificados e soluções. Observou-se dados inconsistentes e duplicados gerando ambiguidade. Durante o tempo foram inseridos muitas dimensões ao dado das ordens de serviço que dificultou o preenchimento e criaram-se informações dúbias e com pouca utilidade, que poderiam se obtidas com tratamento de dados somente e não com inserções de mais campos de dados ao sistema. Constatou-se que o dado puro das ordens de serviço foi insuficiente para a construção de um processo de melhoria no atendimento dos incidentes em relação a redução de tempo. Uma limpeza e integração de dados com outras bases seria fundamental para melhorar a qualidade do dado das ordens de serviço.

3.6 *Análise dos Registros dos Clientes*

Os registros dos clientes foram analisados para um melhor entendimento do problema. O fator motivador era que as ordens de serviço em si eram insuficientes para sabermos qual foi realmente o problema e a sua solução. Esta análise foi feita em paralelo com a análise das ordens de serviço (Figura 5) e foram usados artefatos de coleta das informações das bases dos clientes. Dados como o número de transferências entre os times de suporte eram mostrados em detalhes, bem como datas de início e término do problema mais precisas. Observou-se uma carência de dados nas ordens de serviço em relação ao detalhamento do tempo total de atendimento. Neste ponto o registro do cliente ajudou a mapear este detalhamento de tempo para planejar os experimentos que seriam realizados. Estes dados foram extraídos de ferramentas de controle do cliente como ManageNow e ServiceNow.

3.7 *Análise sobre a necessidade de classificação de incidentes*

Observando a Figura 5, a partir da análise das ordens de serviço e a constatação de que os dados puros do atendimento do sistema seriam insuficiente para definir um banco de dados de soluções, definiu-se como premissa importante a necessidade de classificar os dados de forma que uma transformação e integração com outras fontes de dados pudesse melhorar a qualidade do dado. A partir da revisão bibliográfica observou-se que o ODC poderia ser uma alternativa promissora devido ao fato que o ODC possui o objetivo de melhoria do processo. Porém o ODC na sua forma original tem uma conotação muito voltada a outras áreas do desenvolvimento e apesar de ser simples para uma definição inicial do problema feita pelo primeiro nível de suporte (a mesa de atendimento) não seria suficiente para resolver problemas de ambiguidade na classificação. Estas ambiguidades precisariam ser sanadas para termos uma base de dados consistente de soluções com alta granularidade. Para resolver o problema de granularidade foi concebido uma extensão do ODC utilizando o conceito de componentes de serviço que seriam componentes como memória, CPU, bancos de dados, middlewares, diversos itens de configuração de ambientes que seriam acoplados ao modelo ODC original.

3.8 *Análise sobre a necessidade de agrupamento de soluções*

Analisando as ordens de serviço e o atendimento aos incidentes, notou-se uma grande falta de informações para resolver os atendimentos. Foi observado esta necessidade a partir da

classificação (Figura 5) e a necessidade de agrupamento de soluções em torno da classificação definida. As ordens de serviço em problemas parecidos possuiam muitas soluções muito diferentes. As soluções muito diferentes ocasionavam um aumento de tempo no atendimento em alguns casos analisados. Um agrupamentos de soluções em torno de uma classe de problemas poderia resolver o problema. Foi pensado num mecanismo que a partir deste agrupamento pudesse de alguma forma mostrar as melhores soluções. Foi pensado numa forma de priorização de soluções que tivessem em mais uso. E este uso poderia ser alterado conforme o aumento de informações coletadas diariamente por mecanismos de alimentação da base de dados.

3.9 *Definição de complexidade de Incidentes*

A partir do agrupamento de soluções (Figura 5), obsevou-se a partir do trabalho no dia a dia junto ao time de suporte a necessidade uma transferência mais correta entre times de suporte. Um ponto importante de melhoria do processo atual seria definição de complexidade dos incidentes. Esta definição seria fundamental para a transferência correta de incidentes aos times de suporte, podendo ser investigada uma redução de tempo de atendimento a partir de um mecanismo eficiente de transferência. Foi pensando numa forma de como definir este mecanismo que fosse confiável de forma que problemas mais complexos pudessem ser transferidos corretamente para times com mais conhecimento e menos complexos para times com menos conhecimento. Uma alternativa seria associar de alguma maneira ao modelo ODC que na etapa de classificação foi o escolhido para ser usado. Transferências equivocadas para times em algumas amostras revelaram um aumento no tempo de atendimento, sendo portanto um campo interessante a explorar. Baseados em algumas pesquisas a partir revisão bibliográfica chegou-se a o estudo do gatilho, porque o gatilho é justamente um campo do ODC definido na abertura de um defeito.

3.10 *Definição das Amostras*

Observando a Figura 5 tem-se a fase de definição das amostras que foi implementada por meio de uma pesquisa descritiva de 2 conjuntos de amostras. No primeiro conjunto foram 2 amostras de 50 ordens de serviço de uma Empresa Cliente “X” gerenciada pelo provedor de serviços (Empresa “A”), sendo uma amostra relativa ao mês de Abril e outra relativa ao mês de Outubro de 2013. O segundo conjunto foi extraído ordens de serviço de uma Empresa

Cliente “Y” também com 2 amostras de 50 ordens de serviço para os mesmos períodos (Abril e Outubro de 2013).

A distância de tempo entre as amostras (Abril e Outubro) foram adotados para que fosse possível aplicar uma simulação de diminuição do tempo de atendimento com o método proposto, mas sem interferência entre as amostras. Essa interferência poderia ser, por exemplo, de chamados parecidos, como erros em cascata de um mês anterior para um outro subsequente.

3.11 *Definição de Experimentos Manuais*

A definição de experimentos manuais, conforme observado na Figura 5, surgiu pela necessidade de validar um processo proposto com relação a redução de tempo de atendimento junto aos incidentes (resolução de incidentes e transferência de incidentes entre os times de suporte) . Este experimento seria a partir de uma classificação manual, agrupamento de soluções manual e definição manual de complexidade. Esta redução de tempo seria uma estimativa de redução comparando-se com o processo atual. Uma aplicação real em produção junto aos times de suporte só seria possível com o experimento manual para analisar a viabilidade do novo processo.

3.12 *Análise de Interferência entre Classificação/Agrupamento e reduçao de tempo de atendimento aos incidentes*

Nesta fase, observada na Figura 5, foi feita uma análise entre a condição do agente (Classificação e Agrupamento de soluções) e condições do efeito (Diminuição do tempo de atendimento) através de uma simulação, com o processo existente e o processo proposto. Esta análise seria para verificar através de estimativas se as soluções agrupadas poderiam reduzir o tempo de atendimento dos incidentes através das amostras definidas na etapa anterior. Em cada elemento (ordem de serviço) do conjunto da amostras foi feito a entrada do agente (Classificação e agrupamento) no processo proposto e seria verificado se o efeito (Diminuição do tempo de atendimento) ocorreu, sendo feita esta verificação de forma manual.

3.13 *Análise de Interferência entre Definição de Complexidade e redução de tempo de atendimento aos incidentes*

Nesta fase, observada também na Figura 5, foi feita uma análise entre a condiçao de um segundo agente (Definição de complexidade) para transferir o chamado e condições do

efeito (Diminuição do tempo de atendimento) através de uma simulação com o processo existente e o processo proposto. A definição da complexidade teria como um dos resultados o time que seria transferido o chamado de suporte dentro do método proposto. O Efeito medido seria se a Diminuição do tempo de atendimento ocorreu junto a cada ordem de serviço de cada conjunto de amostras, sendo feita esta verificação de forma manual.

3.14 *Definição de Experimentos Automatizados*

Ainda na Figura 5, foi pensado em como o processo poderia ser utilizado em larga escala surgindo a idéia de se elaborar experimentos automatizados. Estes experimentos utilizando alguns artefatos em data mining e um banco de dados poderiam simular a geração de um data warehouse permitindo uma viabilidade de automatizaçao. Junto a isto o design de uma ferramenta com uma interface amigável de acesso ao data warehouse foi elaborada.

3.15 *Procedimentos Específicos*

Para a análise de interferência foram usadas planilhas eletrônicas no formato xls sendo feito manualmente cada análise de ordens de serviço dentro das planilhas. Para o experimento automatizado foi usado o classificador Weka (WEKA, 2013) para a mineração de dados e a ferramenta jMYSTIQ. (JMYSTIQ, 2013) para o carregamento dos dados para o banco de dados. Para a persistência dos dados foi utilizado o Sistema Gerenciador de Banco de Dados DB2 Express-C, ferramenta gratuita na versão 9.7 fix pack 4. (IBM DB2 Express-C, 2012). Para a modelagem de dados foi utilizado o MySQL WorkBench (ORACLE MySQL WorkBench, 2012).

4 PROCESSO PROPOSTO

O processo proposto para este trabalho verifica a possibilidade de redução de tempo no atendimento dos incidentes dentro do processo atual de atendimento e custo devido ao fato que menos recursos técnicos podem ser envolvidos na resolução do problema. O processo é dividido em 2 fases. A fase de ***Data Warehousing*** na qual o objetivo seria a construção do data warehouse e a fase de pós processamento ou ***Warehouse DBMS*** que seria a fase do uso do data warehouse. (HAN; KAMBER; PEI, 2012).

A fase de *Data Warehousing* estaria dividida em 5 etapas principais de acordo com a Figura 6:

1. Extrair, carregar e integrar as ordens de serviço do cliente (Pré-Processamento)
2. Transformar os dados iniciais em atributos ODC relevantes
3. Executar o mecanismo de detalhamento dos atributos ODC – Drill Down/Slice
4. Classificar em Bug determinístico e não determinístico
5. Construir o Data Warehousing de decisões de suporte

As etapas 2 , 3 e 4 fazem parte da tarefa de classificação em que temos processamento OLAP (Online Analytical Processing). Veja Figura 6.

A fase de pós-processamento ou *Warehouse DBMS* é composta das sequintes etapas (Figura 6):

1. Pesquisar por uma ou mais chaves
2. Obter o time correto que recebera o chamado de suporte
3. Obter as melhores solução para o tipo de problema descrito

Cabe ressaltar que o processo atual de atendimento não é alterado , seria alterado a forma como os recursos humanos seriam acionados no novo processo. Estas fases e etapas serão detalhadas neste capítulo sendo a sua visão geral, pode-se visualizar no diagrama de atividades da Figura 6.

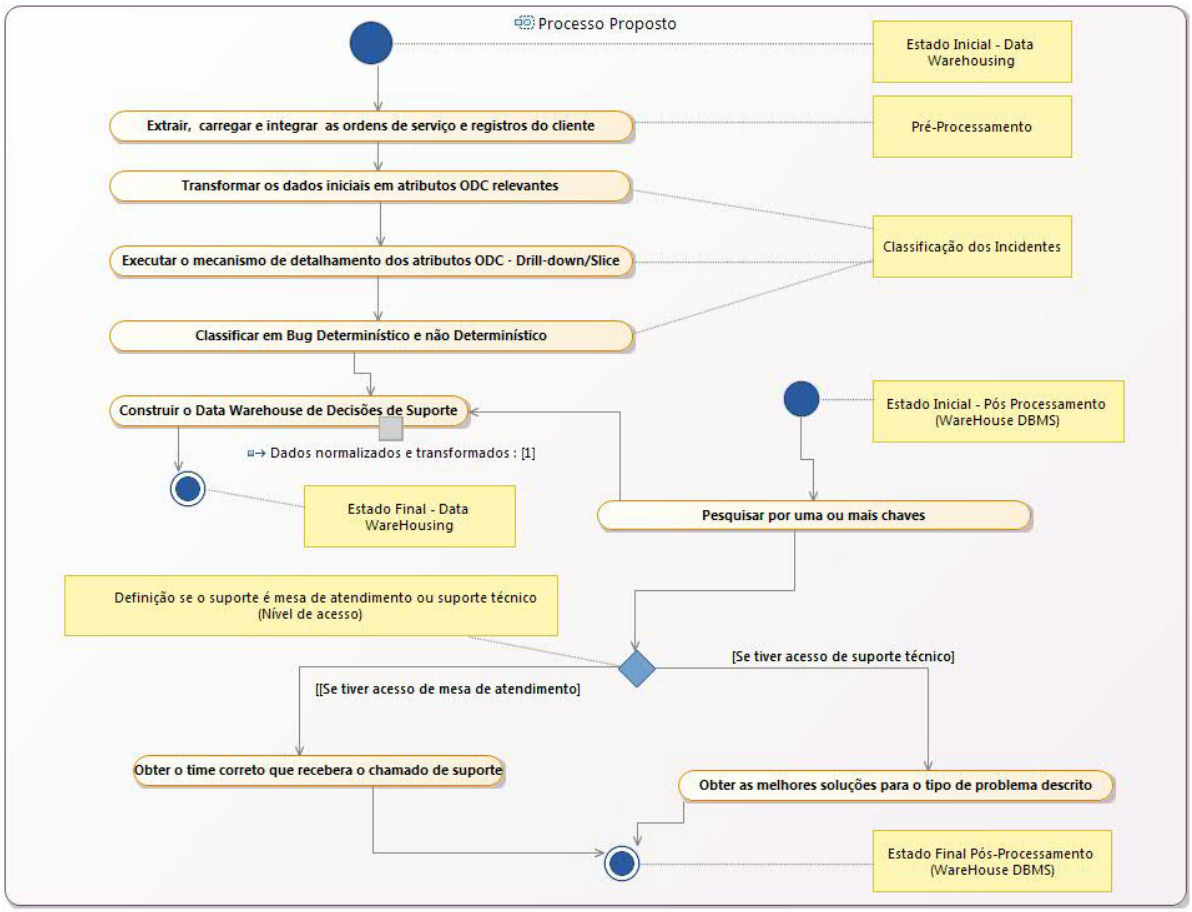

Figura 6 - Diagrama de atividades para o processo proposto

O processo proposto foi originado a partir de algumas etapadas da pesquisa como a análise de classificação de incidentes, análise de agrupamento de soluções e definição de complexidade de incidentes.

Observando o diagrama de atividades da Figura 6, verifica-se as fases de pre-processamento e de pós processamento. A classificação de defeitos faz parte do pré-processamento. A parte de direcionamento aos times de suporte e o ranqueamento das melhores soluções para uma classe de problemas, na qual o incidente foi definido, fazem parte da parte do pós-processamento.

4.1 *Extrair, carregar e integrar as ordens de serviço e registros do cliente*

Conforme observado no método descrito no Capítulo 3, os dados precisaram ser obtidos de mais de uma fonte de dados. A fase da extração e carregamento seria a obtenção dos dados de planilhas eletrônicas no caso das ordens de serviço e base de dados que seria o registro dos incidentes a partir da visão do cliente. Nesta fase a limpeza de dados seria

utilizada para eliminar dados incompletos ou consolidá-los a partir das informações das duas fontes de dados. Estas fontes de dados seriam relacionadas pelo número do incidente.

4.1.1 Transformar os dados iniciais em atributos ODC relevantes

Nesta fase é aplicado o processo de ETL, desmembrando os dados iniciais das ordens de serviço para os atributos ODC importantes para uma análise de causa raiz. Veja Figura 7.

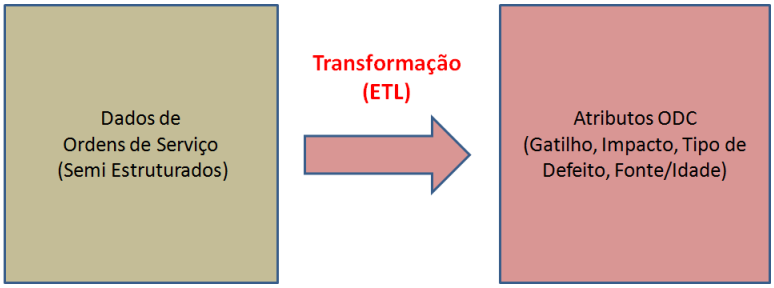

Figura 7 - Processo proposto de ETL entre os dados semi-estruturados das ordens de serviço e atributos ODC de análise de causa raiz

Analisando a Figura 7, do lado direito, estes atributos do ODC selecionados, dentro do escopo da classificação de incidentes são os seguintes: o Gatilho ODC e Impacto ODC, quando um defeito é encontrado e quando um defeito é consertado o tipo de defeito ODC e a fonte do defeito ODC. Esta relação é apresentada na Figura 8. Estes atributos foram selecionados para uma multidimensionalização dos dados de suporte, além de auxiliar a definir com mais exatidão um defeito.

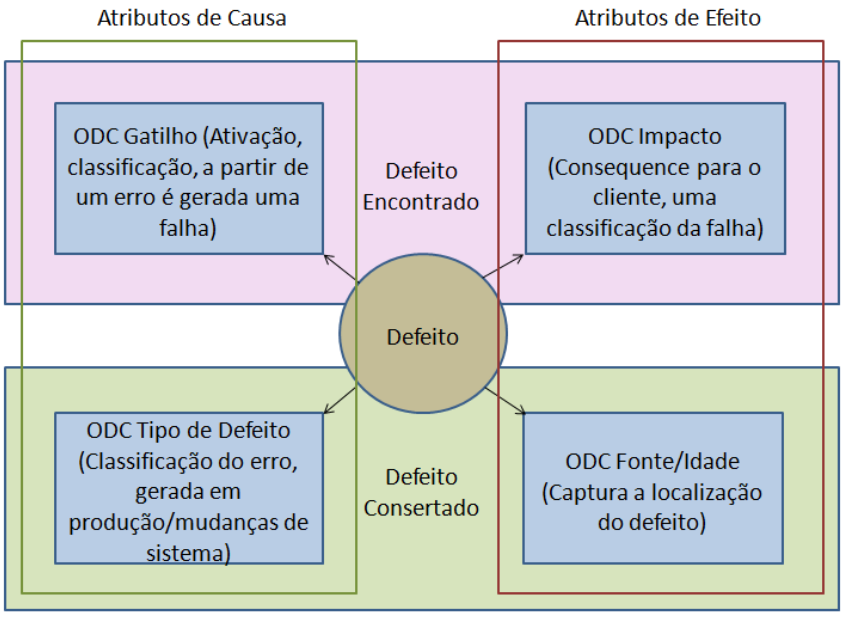

Figura 8 - Atributos ODC (Gatilho, Impacto, Tipo de Defeito, Fonte/Idade) relavantes para a multidimensionalidade dos dados de defeitos (CHILLAREGE, 2006)

Todos os atributos da Figura 8 estão relacionados a um defeito. Relativamente ao defeito encontrado, o gatilho ODC é um atributo importante ao processo de classificação proposto, pois ele auxilia na definição do tipo de *bug* (Bug determinístico e não determinístico). O impacto representa a classificação da falha na visão do cliente (CHILLAREGE, 2011). Relativamente ao defeito consertado, o tipo de defeito mede o processo de desenvolvimento de software onde os defeitos são consertados a respeito das falhas, representando qual correção foi aplicada (IBM RESEARCH, 2013). Por fim o ODC fonte, representa a localização do defeito (CHILLAREGE, 2006).

Outra relação importante na Figura 8 é a relação de causa e efeito. Os dois atributos à esquerda estão relacionados a causa. A causa, por sua vez, se relaciona ao desenvolvimento de software, teste e consequentemente, campo (CHILLAREGE, 2006). O gatilho traz uma falta a superfície (Falha). O defeito consiste na correçào da causa que originou o problema. Um programador que faz a correção escolhe o tipo de defeito usualmente para classificá-lo e a seleção do tipo de defeito implica numa eventual correção (CHILLAREGE, 1992).

Ainda na Figura 8, os dois atributos no lado direito são associados com o efeito. O efeito diz respeito ao cliente. Cada defeito cria um ligação entre os aspetos associados a causa com o efeito. Quando olha-se um grupo de defeitos, este dado contrói-se um modelo que relaciona a causa para o efeito (CHILLAREGE, 2006).

Analisando o dado fonte para esta transformação de acordo com a Figura 7, partimos do princípio que os dados iniciais são semi-estruturados. Realizando um estudo das fontes de dados podemos encontrar estes dados em um formato mais elaborado, com uma divisão maior de campos de dados, permitindo um desmembramento para as variáveis ODC mais simples. Por outro lado foram analisadas fontes de dados com poucos campos com mais texto de conteúdo. Neste caso um desmembramento seria mais complexo como mostrado na Figura 9.

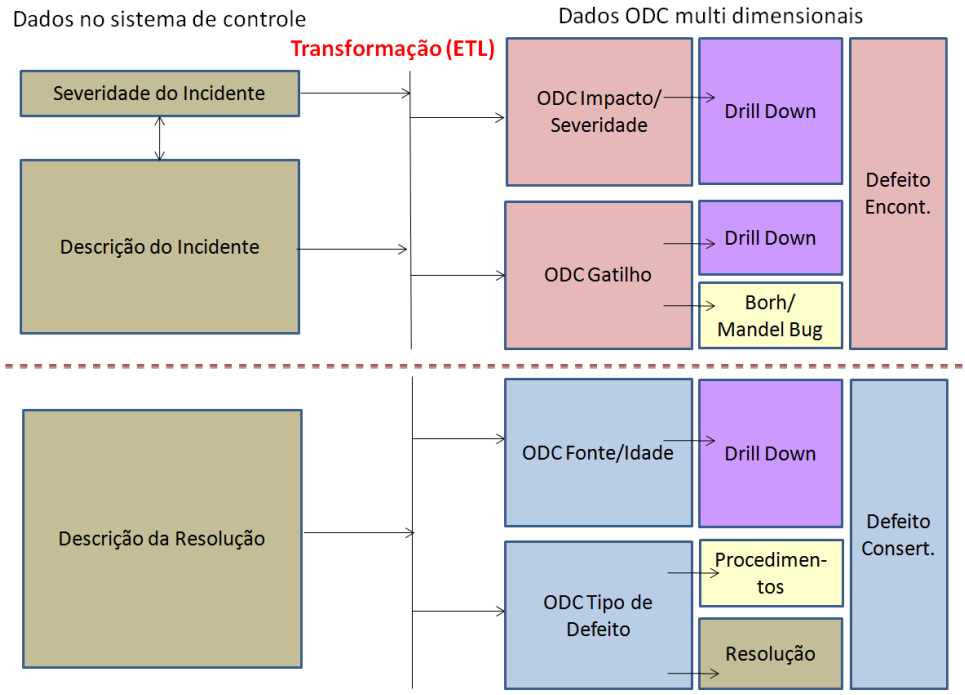

Figura 9 - Desmembramento proposto das variáveis operacionais para as variáveis ODC (Impacto/Severidade, Gatilho, Fonte/Idade e Tipo de Defeito).

A complexidade em questão é maior porque há menos campos específicos, determinando mecanismos de data mining são necessários para obter a partir de textos maiores as informações necessárias para se obter os campos ODC (Gatilho, Impacto, Tipo de Defeito e Fonte/Idade). Em alguns casos pode acontecer da severidade estar descrita em texto dentro do campo da descrição do chamado, diminuindo em 2 campos de dados. A figura 9 mostra outros desmembramentos a partir das variáveis ODC e que é explicado em outras seções do processo proposto.

Um desmembramento mais simples, porque há campos de dados mais específicos é mostrado na Figura 10.

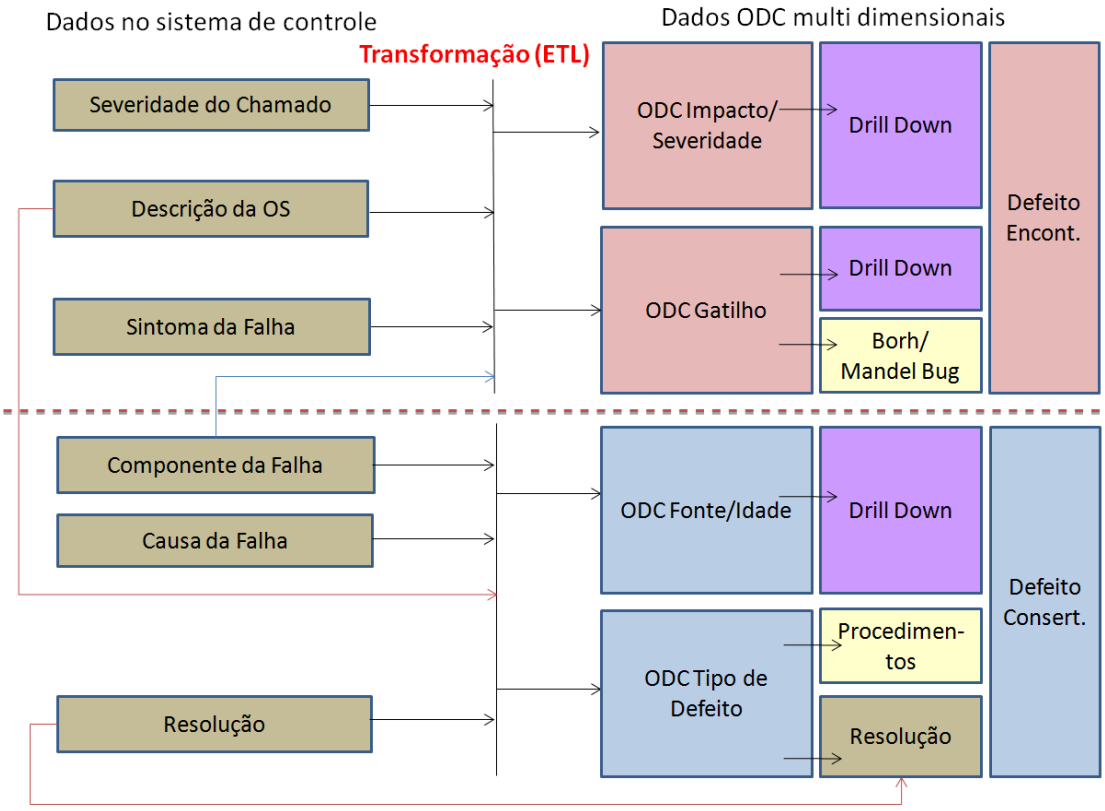

Figura 10 - Desmembramento proposto das variáveis operacionais para as variáveis ODC com maior número de campos específicos

. Inicialmente, o dado é classificado manualmente baseado no ODC adaptado com parâmetros para a manutenção de software por detalhamento (*drill-down*) e classificação *Bohr-Mandel* (CHILLAREGE, 2011) como apresentado na Figura 10.

Ainda na Figura 10, quando o defeito é encontrado, a severidade do chamado, a descrição da ordem de serviço (OS) e o sintoma da falha são mapeados para o ODC Impacto/Severidade. O Gatilho ODC é mapeado a partir da descrição da ordem de serviço, sintoma da falha e do componente da falha. Todo este mapeamento é feito manualmente através de experts no domínio de conhecimento. O produto final desta fase é um conjunto de treinamento que é usado em um processo automático utilizando mineração de dados. Quando um defeito é consertado, o componente da falha, a causa da falha e a resolução são mapeados para o ODC Fonte/Idade e a resolução é mapeada para o tipo de defeito.

4.1.2 Drill-down

O processo inicial de extração de dados aplica operações utilizadas em *Data Warehousing,* chamadas de *drill-down/roll-up,* juntamente com *data slicing*. O *drill-down* é um processo que divide uma área de informação, aumentando o detalhamento dos dados, Roll-up é a operação inversa ao *drill-down*. O termo fatiar os dados ou *data slicing,* na análise de dados, implica em uma redução de um conjunto de dados para subconjuntos. Estes conceitos foram explicados detalhadamente na seção 2.4.1 – Operações Básicas, neste trabaho. Os dados ODC selecionados para multidimensionalidade, como apresentado na seção anterior, são descritos nos atributos **impacto, gatilho, fonte e tipo de defeito**. Estes atributos do ODC descrevem melhor o defeito segundo Chillarege (2006). Este método de usar o *drill-down* com o fatiamento de dados pode ser combinado com uma análise de causa raiz clássica em pequenos conjuntos de dados (CHILLAREGE, 2006). Isto porque especificando melhor o defeito pode-se chegar a uma resolução mais rápida usando métodos de análise de causa raiz como os cinco porquês(CHILLAREGE, 2006).

Este método é chamado ODC-SC ou ODC *Service Components* sendo descrito em Manhães, Emer e Bastos (2014), o qual explica a correlação entre gatilhos e componentes de serviço. Nessa correlação, o gatilho no ODC representa o ambiente ou condição em que o defeito foi encontrado, como, por exemplo, Stress ou Sobrecarga (Workload/Stress), mas, para um melhor diagnóstico do defeito, é necessário correlacionar o defeito com qual dos componentes de serviço o defeito foi encontrado, como, por exemplo, com a Memória do Sistema. Somente indicando Sobrecarga ou Stress fica muito difícil prover uma solução mais precisa para este problema, ou seja, sem o apoio do componente de serviço (Memória do Sistema).

Componentes de serviço representam importantes camadas de softwares e hardwares em um provedor de serviço como, por exempo, bases de dados, *middlewares*, rede, CPU, infraestrutura de segurança, armazenamento (*storage*), sistemas operacionais e outros componentes.

O julgamento, sobre qual componente de serviço será selecionado, é construído a partir dos dados fonte, tais como, a descrição da ordem de serviço, o componente da falha, ou mesmo o sintoma da falha quando da abertura do defeito, conforme apresentado na Figura 11. Um exemplo seria que em uma descrição do problema apresentar alguma informação do componente que deu o problema como uma base de dados corrompida. O componente da

falha quando preenchido ja mostra esta informação necessária ao detalhamento, bem como o sintoma da falha. Esta análise seria feita manualmente com a experiência de experts no assunto.

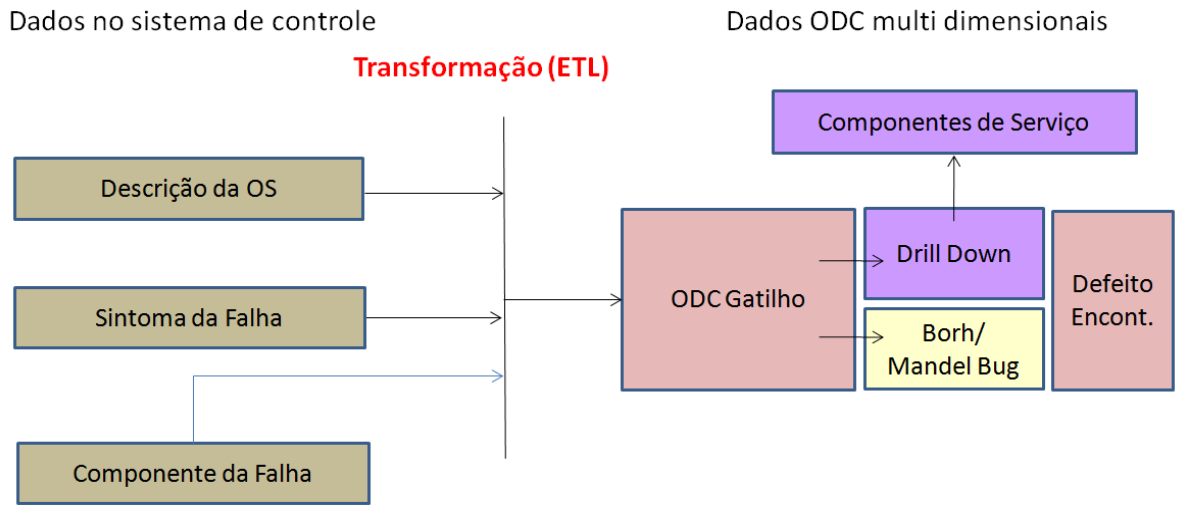

Figura 11 - Mecanismo de drill-down proposto para o atributo gatilho ODC - Visão geral de mapeamento

Nos dados históricos de defeitos, o primeiro passo é classificar os parâmetros quando o defeito for encontrado (gatilho ODC e impacto ODC). Após isto, se aplica a operação de *drill-down* no gatilho, para se obter o componente de serviço. Quando um novo defeito vem do primeiro nível de suporte, numa condição real, é normal prover uma classificação geral para o componente de serviço, e esta classificação, muitas vezes, dentro do componente da falha, não é muito precisa, pois o primeiro nível de atendimento não tem muito conhecimento de detalhes técnicos a respeito do sistema. Mas, neste primeiro nível de classificação, pode-se prover uma definição genérica como um problema de *middleware*, ou de base de dados, por exemplo. A Figura 12 apresenta esta primeira classificação.

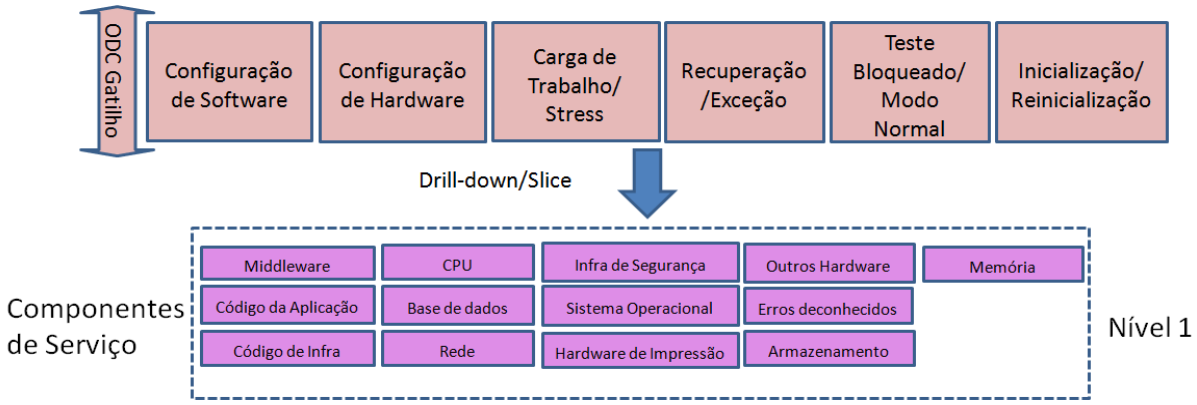

Figura 12 - Mecanismo de drill-down proposto para o atributo ODC Gatilho – Primeiro nível de detalhamento

Considere, como exemplo, um histórico de defeitos no qual uma aplicação *web* alocou uma alta quantidade de memória, originando um problema de performance. Neste caso, a partir da Figura 9, é possível verificar que a classificação mais apropriada seria a Carga de Trabalho/Stress para o ODC Gatilho e, de acordo com os dados de descrição do incidente, utilizando-se a *operação de drill-down*, o *Middleware* seria o Componente de Serviço (Nível 1 da Figura 9).

O Impacto ODC, conforme explicado na seção 2.3.1 (Abertura de um defeito), refere-se a aplicação de negócios impactada e, no exemplo descrito no parágrafo anterior, seria a Performance.

O próximo nível de classificação diz respeito quando um defeito é consertado (Figura 13). Neste ponto, os atributos Fonte ODC e Tipo de Defeito ODC são detalhados a partir da classificação inicial padrão ja feita pelo modelo ODC, que por sua vez é feita manualmente ou pelo menos elaborado a partir de um conjunto de treinamento por experts no assunto. Utilizando as informações coletadas na abertura do defeito para o Gatilho ODC, é estabelecido um nível mais detalhado para o atributo Fonte ODC, conforme apresentado no Nível 2, da Figura 13.

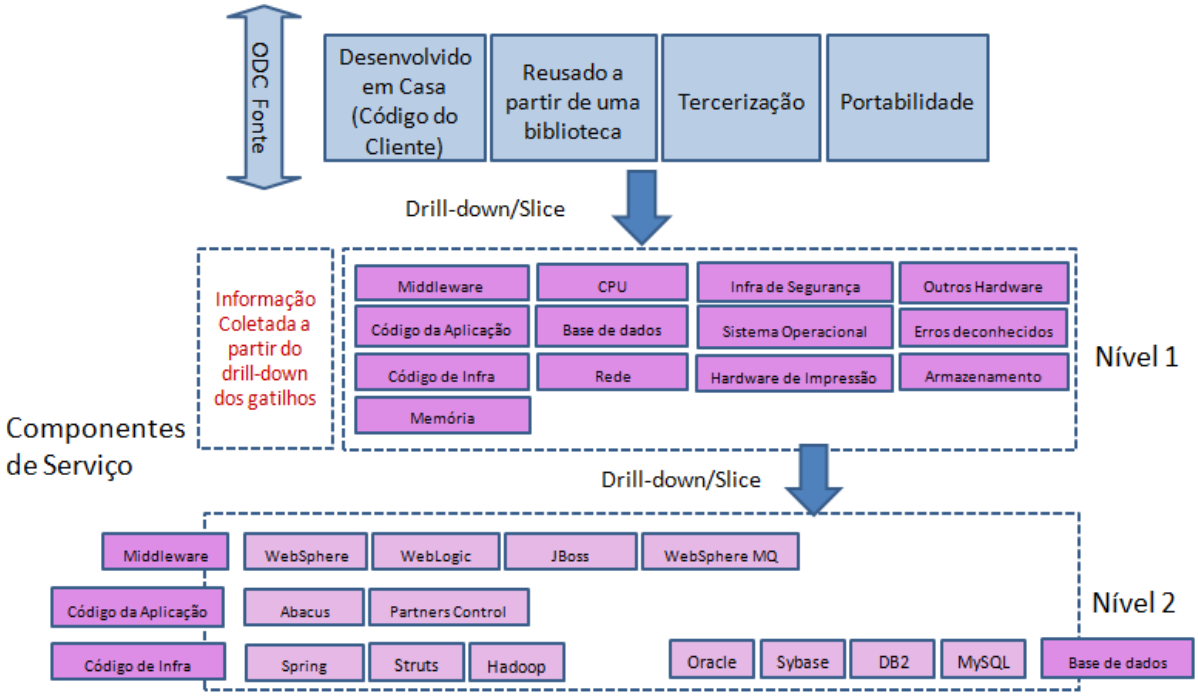

Figura 13 - Drill-down proposto do atributo fonte ODC usando o primeiro nível de ODC gatilho

No Nível 2 (ver Figura 13) é descrito em profundidade onde pode ser achado o problema e onde o mesmo foi consertado a partir do Nível 1 (Visão mais superficial do componente no qual o defeito surgiu a superfície). Pode-se notar que o primeiro detalhamento é obtido a partir do Gatilho ODC quando o defeito foi encontrado. Além disso, um ponto importante é posicionar os componentes de serviço na fundação do Fonte ODC. As classificações padrão de valores para o atributo Fonte ODC seriam as seguintes (IBM RESEARCH, 2013) :

- Desenvolvido em casa (*Developed In-House*): Quando o defeito foi encontrado em uma área que foi desenvolvido pelo próprio cliente ou organização proprietária do código. Seria em muitos casos um código caseiro, não relacionado a um produto de software, em outras palavras é o código da aplicação cliente.
- Reusado a partir de uma biblioteca: O Defeito é encontrado usando uma parte de uma biblioteca padrão de reuso. O problema poderia ser que a parte reusada foi incorretamente usada ou que existe um problema dentro do código da parte reusada.

- Terceirização: Um defeito está em parte de um software fornecido por um terceiro em relação ao cliente. Neste caso pode ser o código de produto de software.
- Portabilidade: O defeito foi encontrado em um componente que tinha sido validado como correto em um outro ambiente e quando foi feito portabilidade para um novo ambiente o erro surgiu.

Seguindo o mesmo exemplo anterior (Veja Figura 12 e explicações relacionadas), o incidente, que foi classificado como Carga de Trabalho/Stress para o Gatilho ODC e *middleware* como Componente de Serviço (ver o Nível 1 da Figura 13), será classificado como Código da Aplicação (ver o Nível 2 da Figura 13), pois o recurso humano que está resolvendo o incidente sabe a sua causa raiz. De forma semelhante, o recurso humano vai detalhar o Fonte ODC como Desenvolvido em Casa ou *Developed In-House* (parte superior esquerda da Figura 13) e como Código da Aplicação/Abacus (parte esquerda do Nível 2 da Figura 13), que é o nome da aplicação em questão, mudando o Componente de Serviço anteriormente definido como *middleware* quando o defeito foi aberto inicialmente. Um outro nível de detalhamento, sugerido para o Fonte ODC, pode ser uma mensagem de erro, conforme apresentado no Nível 3 da Figura 14.

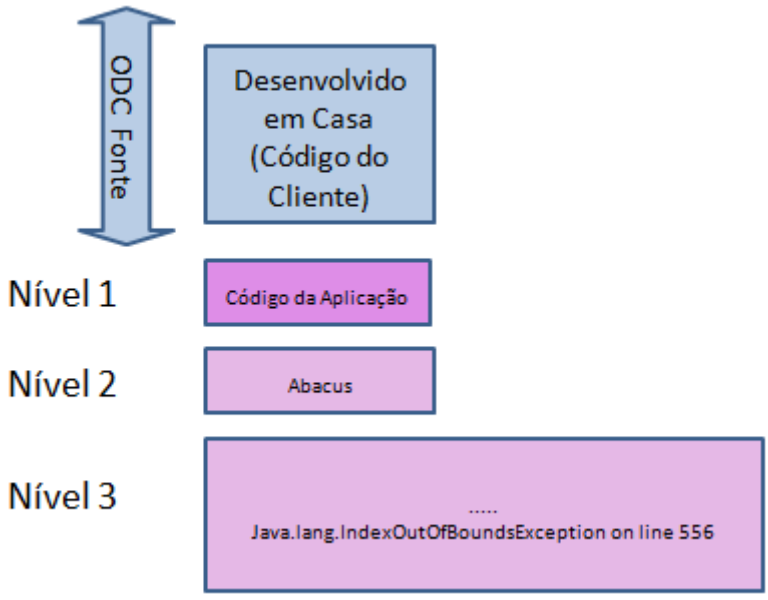

Figura 14 – Nível 3 do ODC fonte proposto pode ser opcionalmente especificado com uma mensagem de erro por exemplo.

A componentização, através destes níveis mais detalhados de Gatilhos e Fontes ODC, garante uma classificação mais específica dos incidentes, podendo-se definir melhor as soluções para os mesmos. Em outras palavras, não é gerado um número muito grande de opções possíveis para um tipo de problema, pois o problema está bem classificado através do componente da falha que foi caracterizado e individualizado.

O último parâmetro para prover a transformação é o Tipo de Defeito ODC, com a resolução do defeito encontrada em algum incidente similar no histórico de incidentes e colocado como subtipo da classificação ODC original do tipo de defeito. Neste ponto, a qualidade da entrada de dados é muito importante e a solução é altamente acoplada a aplicação do negócio.

Seguindo ainda o exemplo anterior (Figura 14), a aplicação Abacus possui procedimentos específicos para reciclar as máquinas virtuais Java, que fazem parte da aplicação, quando um problema como este for encontrado. Um erro nesta reinicialização pode ter um pesado impacto para o negócio porque pode demorar a restaurar o serviço. Portanto além de consertar o problema, em alguns casos são necessários procedimentos específicos de iniciar uma aplicação. Este procedimento pode ser classificado como um subtipo do Tipo de Defeito ODC, conforme apresentado na Figura 15.

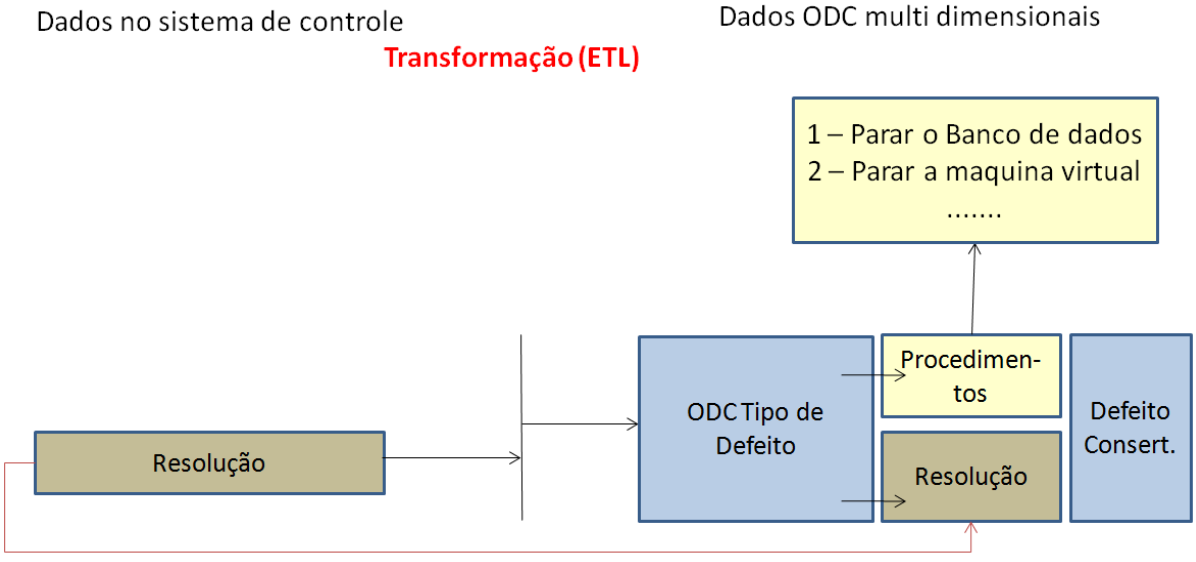

Figura 15 – Procedimento proposto de reinicialização como subtipo do ODC tipo de defeito.

4.1.3 Classificar em Bug Determinístico e não Determinístico

Para se finalizar o processo de classificação de incidentes, definindo uma classificação de complexidade, foi utilizado o conceito de *bugs* determinísticos (*bohrbugs*) e não

determinísticos (*mandelbugs*). Mais detalhes sobre o embasamento teórico pode ser verificado na seção 2.2.3 (Bug determinístico e não determinístico).

O importante nesta pré-classificação, é que os defeitos continuam a ser classificados pelo ODC, somente sendo acrescentando este atributo adicional do tipo de *bug,* ou seja, se é determinístico ou não determinístico, como, por exemplo, um problema reproduzível é um *bug* determinístico.

Analisando a Figura 16, pode-se observar que, a partir do método inicial de transformação de dados entre as ordens de serviço e os atributos ODC, tem-se o mapeamento Borh-Mandel *Bug*. Este mapeamento é realizado, fazendo uma transformação de dados a partir da descriçao da ordem de serviço (dado principal) e o sintoma da falha e componente da falha (campos auxiliares) para o atributo ODC gatilho. Após isto, a partir do artigo de pesquisa descrito em Chillarege (2011) onde temos o mapeamento ODC gatilho e tipos de bug, conseguimos finalizar a definição de complexidade.

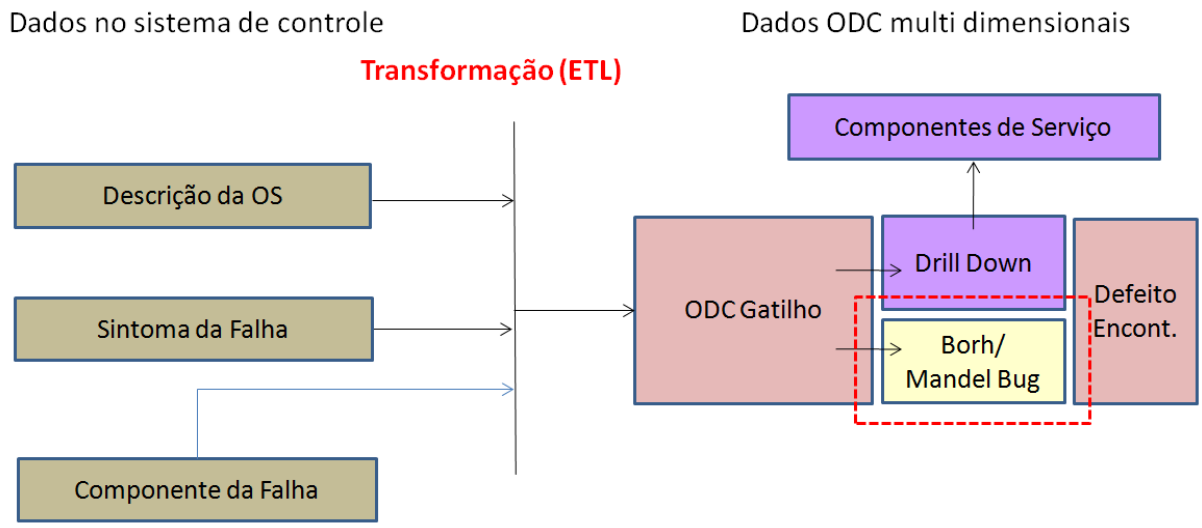

Figura 16 - Mapeamento Bohr-Mandel proposto a partir do ODC Gatilho.

A partir do embasamento teórico definido na seção 2.3.3 (Bug determinístico e não determinístico), os conceitos ali definidos foram ampliados para cobrir todos os tipos de gatilho existentes em produção, fase na qual o sistema já foi entregue e está em operação (ver a Figura 14). Esta ampliação foi feita com a adição de 2 gatilhos de testes de sistema, que são os mais adequados aos testes de campo e que não foram incluídos no estudo de Chillarege (2011). Os novos gatilhos foram os seguintes:

- Teste Bloqueado (em versões anteriores do modelo ODC, também chamado de modo normal): O produto esta operando bem dentro de alguns limites computacionais de recursos e o defeito surge a superfície quando executando os testes de sistema, não podendo ser mostrado em defeitos de suporte vindos do cliente externo (IBM RESEARCH, 2013). Este tipo de gatilho se aproxima mais a um problema não determinístico, não reproduzido facilmente, porque envolve vários outros componentes de hardware e software para a montagem do ambiente quando o software é inserido em um contexto de teste de sistema ou de campo. Em muitos casos demora-se um tempo razoável para se montar o ambiente ou ainda comparar com algum ambiente anterior , por exemplo.
- Inicialização/Reinicialização: Acontece quando o sistema ou subsistema foi inicializado ou reiniciado seguindo algum desligamento anterior ou alguma falha de algum sistema ou subsistema (IBM RESEARCH, 2013). Estes tipos de defeitos em muitas vezes não são muito facilmente isolados (*Mandel Bugs*) devido a dependência do sistema no qual o erro surgiu a superfície com outros sistemas de hardware e software.

Embora a fase de produção ou campo seja a fase relacionada a este trabalho, outros gatilhos podem aparecer nesta fase, partindo da premissa que existem erros que escapam da fase de testes e que aparecem na fase de produção.

Dentro deste contexto de cobrir outras fases, considere um caso no qual o código fonte não foi corretamente testado em fases anteriores do ciclo de desenvolvimento de software, como por exemplo na fase de teste de caixa-preta ou na fase de teste de carga, e, por alguma razão, problemas de concorrência escaparam para a produção (campo). Nesta situação, o defeito escapou da fase correta e causou uma condição de bloqueio (*Lock Condition*). A Figura 17 cobre a situação reportada neste exemplo, significando que uma situação de concorrência reportada como um Mandel-*Bug* e que não faz parte de um gatilho de produção, teste de campo, também é coberta. Isto porque erros de outras fases do ciclo de desenvolvimento podem expandir para a fase de produção.

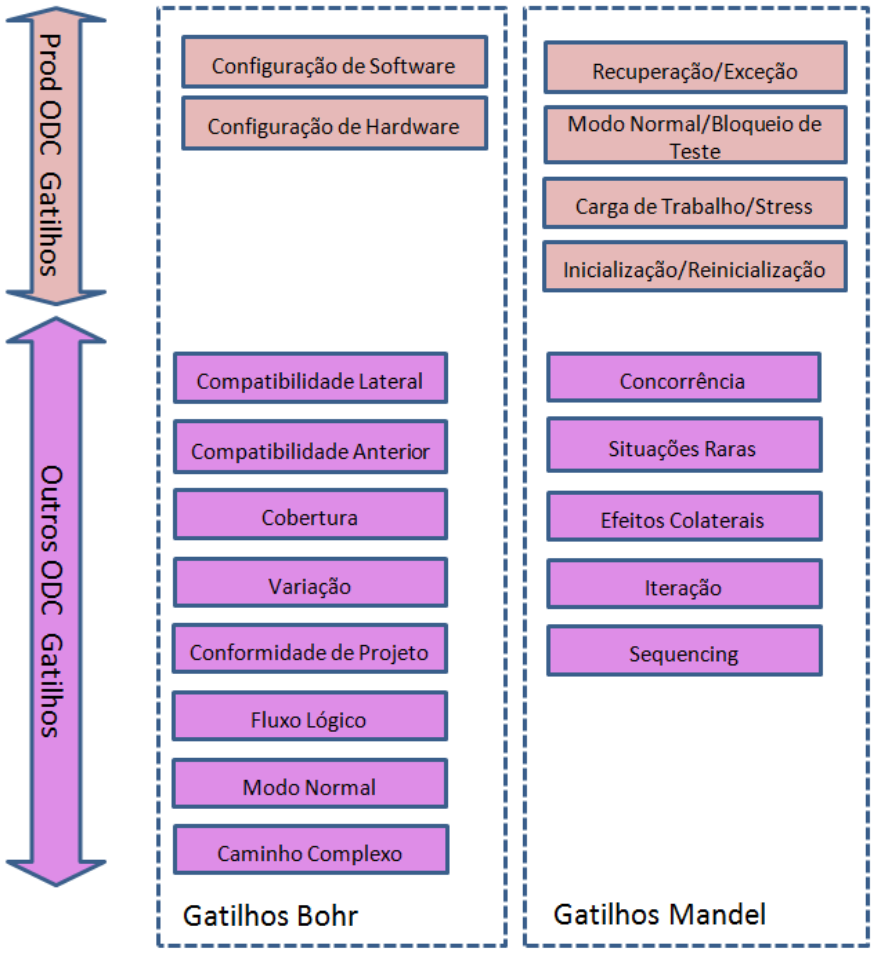

Figura 17 – Distribuição de Bohr-Mandel bugs baseado em Chillarege 2011 e expandida para cobrir todos os testes de campo (Produção).

4.2 *Processo de direcionamento correto para os times de suporte*

Para se definir um processo de classificação de incidentes, foi utilizando o conceito de *bug*s determinísticos (*bohrbugs*) e não determinísticos (*mandelbugs*), onde os *bohrbugs* podem ser resolvidos pelo suporte de primeiro nível e os *mandelbugs* podem ser repassados para os outros níveis mais elevados de suporte, como apresentado na Figura 18, o que poderá levar a uma economia de custos em relação a utilizado de pessoal especializado. Mais detalhes sobre o embasamento teórico pode ser verificado na seção 2.3.3 (*Bug* determinístico e não determinístico).

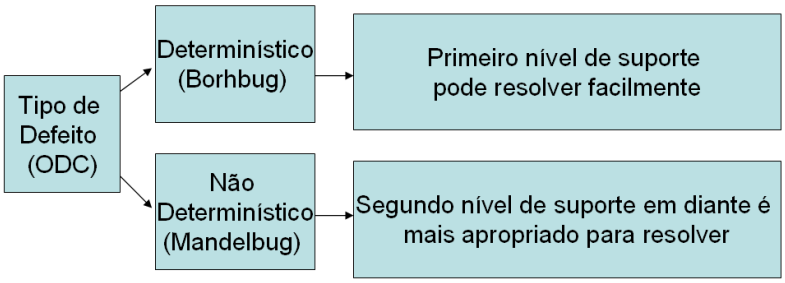

Figura 18 – Definição proposta de tipos de time de suporte apropriados aos tipos iniciais de defeito.

Analisando o processo corrente da Empresa "A", apresentado primeiramente na seção 3.3 (Estratégia de investigação), o mesmo é baseado em ITIL (*Information Technology Infrastructure Library*) e possui vários clientes diferentes com uma considerável complexidade de software. Nesse processo, pode-se observar que recursos humanos com alto custo resolvem problemas simples, sendo que o processo aqui apresentado objetiva reduzir esse custo, propiciando ao primeiro nível de suporte informações mais completas, a fim de que o mesmo repasse o incidente para o time mais apropriado..

4.3 *Melhores soluções para uma classe de problemas*

Na Figura 19 (canto inferior direito), pode-se observar que a Resolução ou os Procedimentos são posicionados como subtipo do atributo ODC Tipo de Defeito. Para priorizar as melhores resoluções para uma classe de problemas é usado o método simples de voto majoritário (HASSOUNA; TAHVILDARI, 2010). O voto majoritário é um método que conta a ocorrência repetida de um valor definido no histórico. Cada ocorrência representa um voto e o valor com mais votos possui um *score* (ranqueamento) melhor. Após isso, uma lista de itens é ordenada, partindo dos itens com o maior número de votos para os itens com o menor número de votos, fazendo um ranqueamento e criando um ranqueamento. Uma variação deste método é o Top-K voto majoritário (HASSOUNA; TAHVILDARI, 2010), onde os limites para o conjunto de dados históricos são limitados a K candidatos com maior pontuação ou *score*. O ranqueamento das melhores soluções é definido com a de maior utilização em primeiro, a segunda em utilização em segundo e assim por diante. Como os dados classificados são incluídos em um *data warehouse* quando há novas ocorrências, o *rank* das melhores soluções é atualizado constantemente. Uma nova solução pode surgir e ser utilizada com mais frequência, ganhando um *score* maior. O *rank* também vale para

procedimentos específicos, que são também igualmente importantes junto às soluções dos problemas, vide Figura 19.

4.4 *Outros atributos coletados para a montagem do data warehouse*

Além dos campos selecionados das ordens de serviço descritos na seção 4.1.1 Transformar os dados iniciais em atributos ODC relevantes, outros atributos das ordens de serviço foram usados, em todos as análises das ordens de serviço,para construir o *data warehouse,* sendo que as razões pelas quais esses atributos adicionais foram inseridos estão indicadas no Quadro 2.

Nome do Atributo	Significado	Motivo
Reported Date Time	Data de abertura da ordem de serviço	Associar características temporais para construir o *data warehouse*
Actual Finish Date Time	Data de Fechamento da Ordem de Serviço	Associar características temporais para construir o *data warehouse*
Fully Qualified Host Name	Localização física (servidor) do incidente (se houver).	Pode ser usado como *drill-down* (detalhamento) do atributo ODC impacto
External Ticket No.	Número de controle de chamado junto ao cliente.	Evitar duplicidades de ordens de serviço para um mesmo chamado do cliente.

Quadro 2 - Quadro explicativo sobre os atributos adicionais a serem coletados das ordens de serviço

Com relação ao atributo "*External Ticket No*", este pode ser uma referência a outras bases de dados do cliente em uma possível integração, como os registros de chamado relativos ao cliente.

4.5 *Opinião de experts*

Outra que pode ocorrer no processo é inserir informações de suporte, dentro dos Procedimentos que fazem parte do ODC Tipo de Defeito (ver a Figura 19), procedimentos específicos criados por experts técnicos. Essa interação com o sistema, serve para refinar procedimentos para *bugs* determinísticos (Bohrbugs), a fim de evitar a chamada de níveis mais altos de suporte.

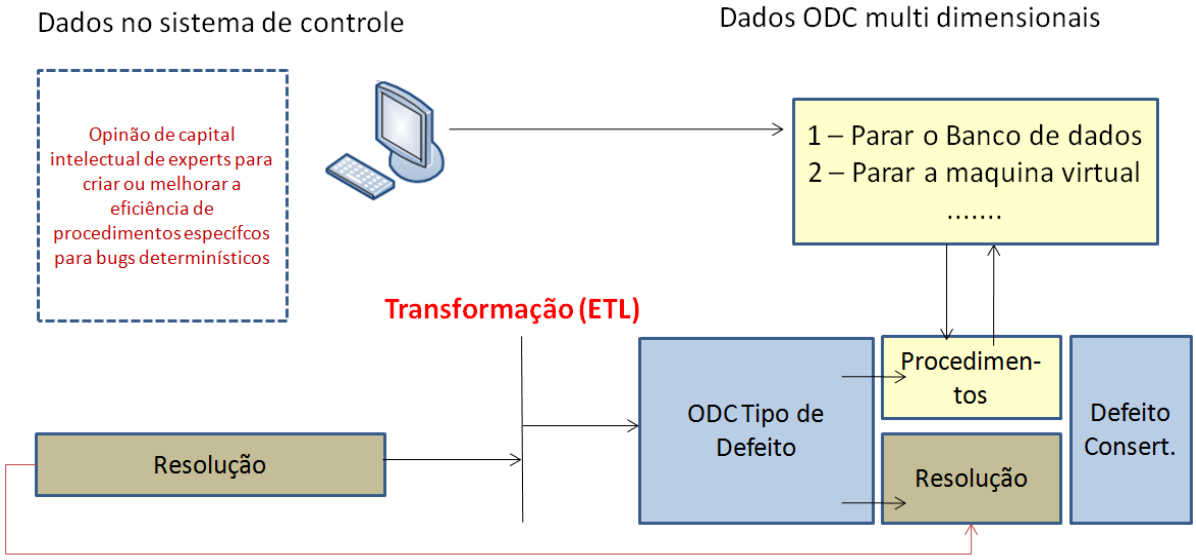

Figura 19 - Tipos de time de suporte apropriado aos tipos iniciais de defeito.

4.6 *Apoio Computacional*

Algumas ferramentas computacionais foram usadas para automatizar o processo. A primeira seria o Weka para automatizar a classificação ODC com as extensões a partir das ordens de serviço em seu formato innicial. A segunda seria o JMYSTIQ. (JMYSTIQ, 2013) para carregar automaticamente as orderns de serviço convertidas para o ODC com as extensões propostas para o data warehousing. Uma alternativa de software livre também foi construída com a linguagem de programação em Java que substituiria o JMYSTIQ, proporcionando uma solução menos dependente de produto específico, mas ainda esta em fase de testes. O próprio banco de dados DB2 Express-C como software livre também foi usado para hospedar o Data warehouse.

A ferramenta computacional final ainda está em vias de desenvolvimento, porém uma versão inicial foi obtida a partir dos experimentos automáticos. Esta versão inicial faz uma simples junção dos artefatos.

5 EXPERIMENTOS DE VALIDAÇÃO DO PROCESSO PROPOSTO

Os experimentos utilizados para validar o processo proposto foram divididos em experimento manual e experimento automático. Os principais objetivos dos experimentos são: verificar se a solução é suportada por clientes diferentes a partir do experimento manual e analisar a possibilidade de ampliar a solução para um volume diário de dados e de consultas utilizando um procedimento automático.

5.1 *Experimentos Manuais*

O Experimento manual segue a definição do método manual explicada na seção 3.11 , em que foi delineado a necessidade deste experimento para uma comparação minuciosa de eficiência de tempo entre o processo proposto e o processo corrente. O experimento executado envolveu o processamento de dois conjuntos de 50 ordens de serviço da Empresa "A", de forma manual, em 2 períodos de tempo (os meses de Abril e Outubro de 2013) para um cliente "X" e dois conjuntos de 50 ordens de serviço do mesmos meses para o cliente "Y". Estes clientes têm ambientes de sistema completamente distintos, apesar de os dois terem contrato de suporte com a Empresa "A".

As ordens de serviço representam um repositório de todo o trabalho realizado para incidentes de suporte de todos os clientes da Empresa "A". Além disso, cada empresa cliente da Empresa "A" tem uma ferramenta de suporte diferente para interagir com os seus próprios clientes. A Empresa "A", quando presta suporte às empresas clientes, utiliza a mesma ferramenta de software, chamada de *Manage Now*, para posicionar as empresas clientes sobre a situação atual do incidente.

O experimento manual, contempla manualmente as 3 fases do processo proposto no capítlo 4 (Pré-processamento,Classificação e Ordenação das Melhores soluções) e é subdividido em :

- Limpeza e agregação de dados.

- Transformação de dados e classificação manual a partir do ODC.
- Aplicação de detalhamento e operações sobre os dados (*Drill-Down/Slice*).
- Definição do tipo de *bug*.
- Ordenação de melhores soluções e sugestão de transferência para o time mais apropriado.

Este procedimento manual foi realizado utilizando-se planilhas eletrônicas. Veja figura 20.

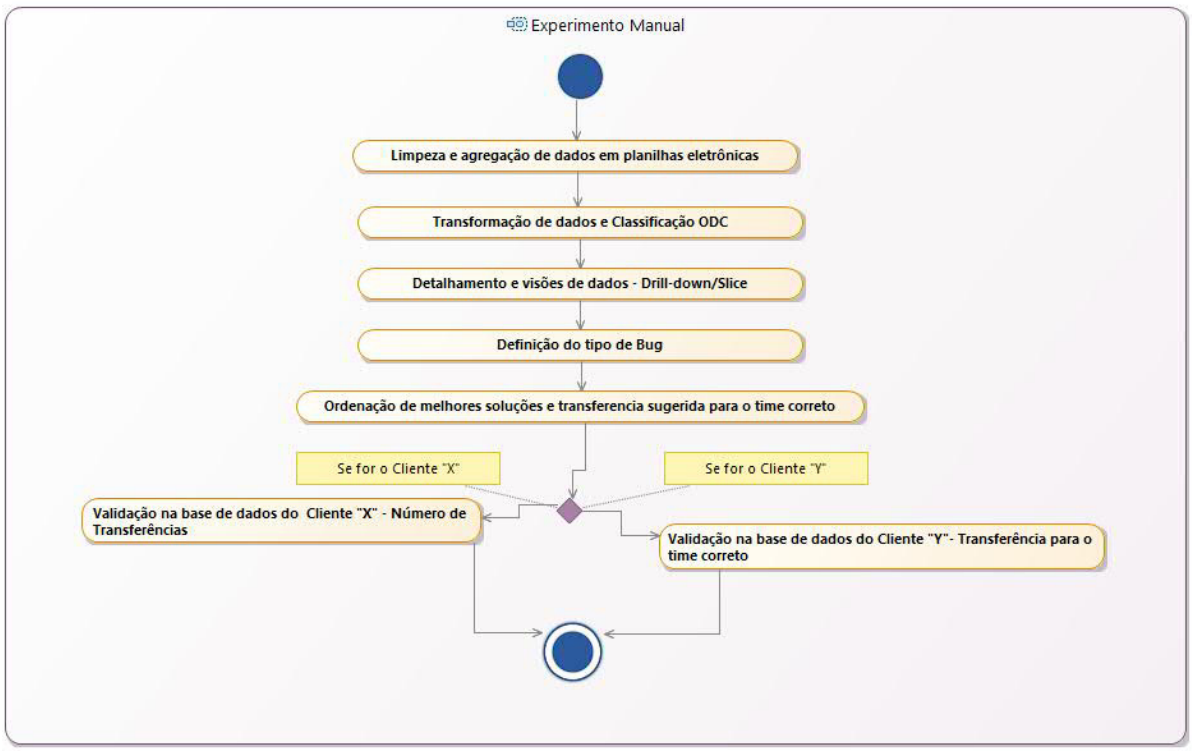

Figura 20 - Atividades do procedimento manual

Durante a limpeza de dados, foram aplicados critérios de exclusão de entradas de dados das amostras. O primeiro critério de exclusão de uma entrada (ordem de serviço) realizado foi se o campo de resolução ou a descrição da ordem de serviço estivesse em branco. Estes campos são fundamentais para identificar qual o problema e a resolução, sem estes ficaria impossível uma análise dos dados.

O segundo critério de exclusão foi se a entrada de dados estivesse com o sintoma da falha e o componente da falha estavam simultaneamente em branco. Caso isso ocorresse, a ordem de serviço seria excluída também, devido ao fato que a classificação ODC seria inviabilizada

Também é verificado o atributo "*External Ticket No*", apresentado na seção 3.8, "Outros atributos coletados para a montagem do data warehouse", que representa o número do chamado do cliente para aquele incidente de software. Um terceiro critério de limpeza de dados para o processo foi excluir entradas que se referissem a esse mesmo número de chamado. Esse fato acontece quando um chamado do cliente passa por vários recursos técnicos dentro de um mesmo time, quando, por exemplo, esse chamado passa de um turno de serviço para outro.

O quarto critério de exclusão foi somente considerar as severidades de nível 1 e 2, as quais são severidades que realmente tem impacto no negócio. Essa classificação é determinada por contrato entre as empresas clientes "X" e "Y" e a empresa prestadora de serviços "A".

Ainda sobre o atributo "*External Ticket No"*, este atributo, que foi adicionado a partir da ordem de serviço, como explicado na seção 3.8, foi utilizado para fazer pesquisas na base dados de controle das empresas cliente "X" e "Y", quando houve a necessidade de se obter mais detalhes de alguns incidentes ou de modificar alguns dados das ordens de serviço quando necessário.

Para a base de dados da Empresa cliente "X", o atributo utilizado para verificar a redução de tempo que o método proposto neste trabalho poderia proporcionar foi o atributo Número de Transferências ou "*Times Reassigned*". Este atributo representa o número de vezes que um incidente é transferido entre os times de suporte. Desta forma, é possível representar indiretamente o tempo gasto para a resolução do incidente, pois quanto mais vezes o incidente é transferido entre os times de suporte da Empresa "A", sem que se chegue a uma solução para o mesmo, maior será o tempo gasto com a resolução do incidente.

Na fase de validação de dados do cliente "X" do experimento manual, conforme Figura 20 (canto inferior esquerdo), foi comparado o número de transferências que ocorreram entre os times de suporte para a resolução do incidente com o número de transferências em uma simulação caso o processo proposto neste trabalho fosse utilizado. Esta simulação passou por todo o processo de suporte, usando os dados formatados manualmente pelas planilhas. A comparação foi realizada a fim de verificar se haveria redução no número de transferências entre os times de suporte e, por conseguinte, redução no tempo de resolução do incidente. Na Figura 21 é apresentada uma representação de parte da planilha utilizada no experimento, representado dados relativos a uma ordem de serviço, incluindo o campo "Com o Novo

Método", que representa o resultado do experimento. Neste caso, comparando-se os campos "Número de Transferências" (49) e "Com o Novo Método" (44), verifica-se que houve uma redução de 5 transferências ou 10%. Isto seria possível esta redução devido a transferência ao correto time de suporte, ou seja, ao invés do time de Messageria seria para o time de websphere. Maiores detalhes das amostras e dos resultados será explicado no capítulo 6 – Resultados.

Ordem de Serviço	W23378966
Descrição	LPPWA1196 LPPWA1196BOLSERVER JMSEXCEPTION: MQJMS2005: FAILED
Numero do Cliente (External Ticket No.)	18500293
Severidade	2
Sintoma da Falha	CONFIGURATION
Componente da Falha	application issue
Solução	provided instructions as requested
Impacto ODC	Reliability
Impacto ODC Componente	BOL
Gatilho ODC	Software Configuration
Gatilho ODC Componente	Middleware
Fonte ODC	Outsourced
Fonte ODC Componente	WebSphere
Tipo de Defeito ODC	Maintenance/Fix Dependencies
Tipo de Bug	B
Número de Tranferências	49
Com o Novo Processo	44
Justificativa	O Recurso engajado no incidente não checou que não era um problema no produto de MQ era no produto websphere. Com o processo proposto seria possível

Figura 21 – Representação de Ordem de serviço utilizada no experimento para a Empresa Cliente "X" com a linha "Com o Novo Processo"

Para a validação dos dados da empresa cliente "Y", foram adicionadas duas colunas na planilha além daquelas definidas para a empresa cliente "X". Na representação gráfica são

duas novas linhas, Time Engajado e Time Correto (Figura 22). A primeira coluna foi adicionada para especificar o time para o qual o incidente foi direcionado primeiramente. A segunda coluna foi adicionada a fim de indicar qual o time correto para o qual o incidente deveria ter sido encaminhado. As colunas foram denominadas como “Time Engajado” e “Time Correto”. O motivo desta validação adicional foi porque o numero de transferências para o Cliente “Y” com o novo processo não diminuíram, exigindo uma verificação mais profunda do incidente ao time no qual foi transferido pela primeira vez. Foi necessário medir a transferência correta entre os times para o cliente “Y”. No cliente “X” não foi necessário porque a redução de transferências foi visualizada com facilidade. A Figura 22 apresenta uma exemplo de entrada para o experimento.

Ordem de Serviço	W17104975
Descrição	RESO Maximo AM - W3SR functinality not working for certain datas
Numero do Cliente (External Ticket No.)	IN3670440
Severidade	1
Sintoma da Falha	APPLICATION SERVICE DOWN
Componente da Falha	W3SR functinality not working for certain datas
Solução	code fix reDeploy via ECR 4808218
Impacto ODC	Capability
Impacto ODC Componente	RESO Maximo AM
Gatilho ODC	Sequencing
Gatilho ODC Componente	Application
Fonte ODC	Developed In House
Fonte ODC Componente	Application Code
Tipo de Defeito ODC	Algorithm/Method
Tipo de Bug	M
Número de Tranferências	1
Com o Novo Processo	1
Justificativa	O time de Administração Web (Time A) foi engajado para resolver o problema. Fez uma parada e inicialização do serviço sem nenhuma investigação adicional
Time Engajado	Time A
Time Correto	Time B

Figura 22 – Ordem de serviço utilizada no experimento para a Empresa Cliente “Y” com as linhas adicionais : “Time Engajado” e “Time Correto”

5.2 *Experimento Automatizado*

O experimento automatizado coberto nesta seção visa emular o processo proposto, a fim de verificar o seu comportamento com um volume de dados normalmente utilizado em sistemas em produção (já entregues). Ele foi dividido em 5 etapas:

1 - Fase manual – Treinamento dos dados.

2 - Fase de automação.

3 - Entrada (Input) final dos dados.

4 - Carga do banco de dados

5 - Artefato de consulta das melhores soluções

A Figura 23 exemplifica o experimento:

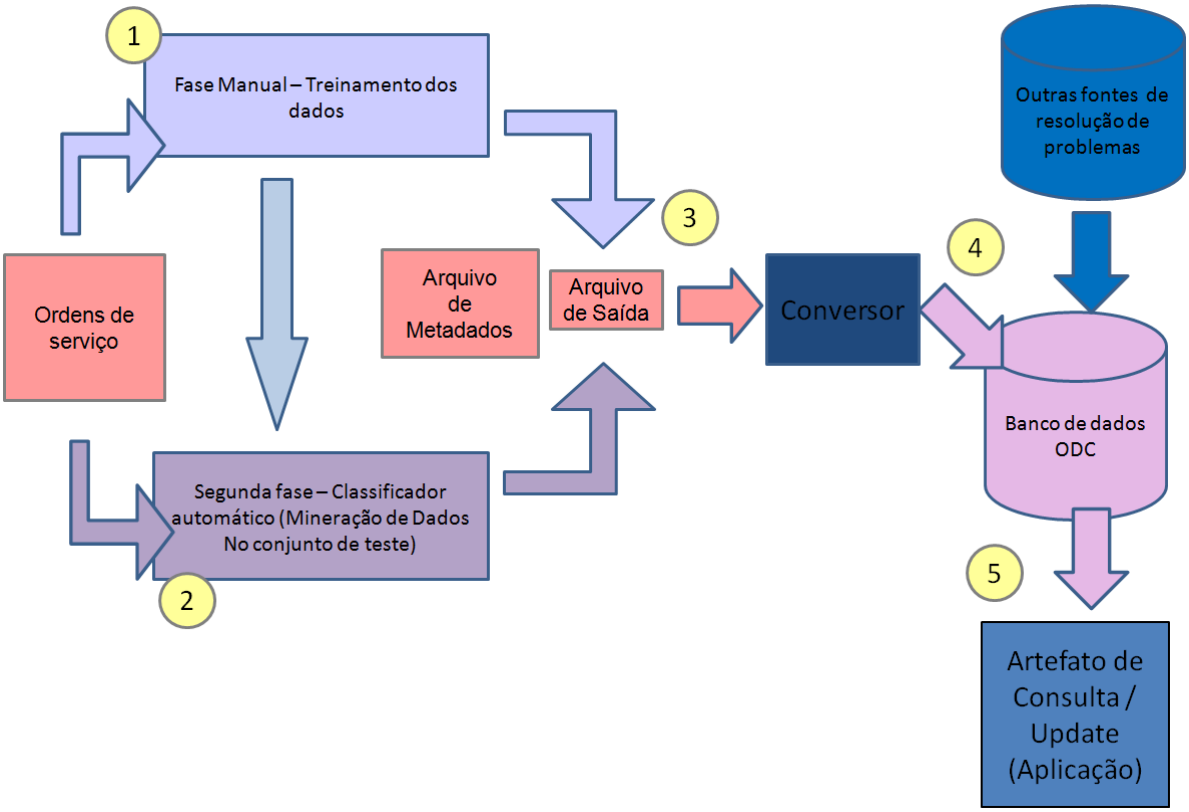

Figura 23 - Modelo de apoio a decisão com as 5 etapas principais.

Na **fase manual** (fase 1 da Figura 23), os dados do experimento foram obtidos a partir dos dados extraídos da ferramenta de chamados de serviço da Empresa "A", que é composta de diversas áreas como banco de dados, *middleware* e e-mail em formato csv (*comma separatue value*). Para esse conjunto de dados inicial, foi feita uma filtragem do cliente "X" e

das severidades de níveis 1 e 2. Estas severidades são as que causam impacto na alta disponibilidade do serviço. Em suma, foi executado manualmente toda a extração, carregamento, limpeza da fase de pré processamento e a parte de transformação, drill-down e classificação do tipo de bug da fase de classificação do processo proposto (Veja Figura 6 ou Capítulo 4). Uma fração deste resultado serviu como um conjunto de treinamento para a próxima fase do experimento automatizado.

Na **fase de automação** (fase 2 da Figura 23), os dados de origem foram extraídos da mesma maneira que na fase manual, utilizando-se o mesmo formato .csv. Esta fase contempla o processo proposto (Extração, Carregamento, Integração, Transformação e Drill-down), Veja Figura 6, da mesma forma que a fase 1 do experimento automático, porém, ao invés de classificá-los no ODC manualmente foi introduzido um classificador de mineração de dados *(Data Mining)*. Três algoritmos foram testados neste experimento: Regras de Associação (WITTEN; FRAN; HALL, 2011), Naive Bayes e Árvore de Decisão (WITTEN; FRAN; HALL, 2011), através da ferramenta Weka (WEKA, 2013). Neste processo, foi necessária a exclusão de caracteres especiais que impediam a leitura do arquivo .csv pelo Weka. Com o conjunto de treinamento obtido da fase manual foi obtido a classificação ODC e o tipo de Bug para o conjunto de testes. A escolha dos algoritmos foi feita a título de experimento utilizando dados das orderns de serviço supervisionados pela fase manual. Não teve um critério específico para a escolha dos algoritmos de mineração de dados.

O ***input* final dos dados** (fase 3 da Figura 23) consistiu em organizar os dados em um formato que pudesse ser carregado no banco de dados. Os dados de entrada utilizados foram o mesmo arquivo .csv (Comma Separated Value), gerado tanto pelo procedimento manual (conjunto de treinamento) como pelo automático (conjunto de testes). Na figura 23 consiste no conversor de dados. Esta fase corresponde a fase do processo proposto de construir o Data warehouse de Decisões de Suporte (Veja Figura 6).

A ferramenta de carregamento dos dados utilizada foi o JMYSTIQ. (JMYSTIQ, 2013). Esta ferramenta é usada para carregar dados de CSVs em banco de dados e é usada em projetos que contemplam dados ODC como ferramenta de trabalho.

Além do arquivo de dados .csv, o JMYSTIQ precisa somente de um arquivo adicional de metadados. Na Figura 24, exemplifica-se o formato do arquivo de metadados. Cabe ressaltar que o campo NumeroOS é importante para a importação pelo banco de dados, por fornecer uma chave primária sem a necessidade de se criar índices no banco de dados.

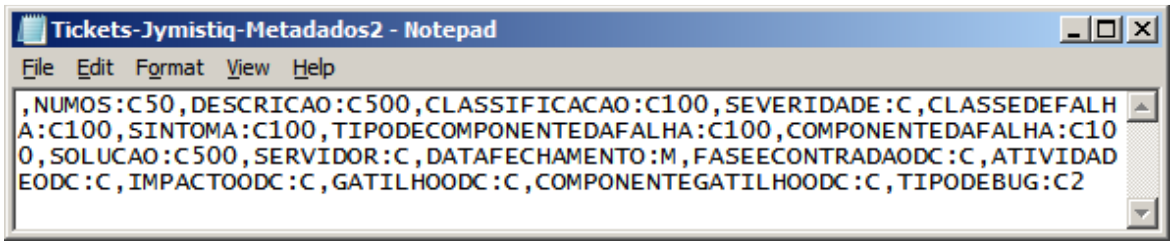

Figura 24 - Arquivo de metadados da ferramenta de gravação no banco de dados

O arquivo exemplo, apresentado na Figura 24, tem o formato começando pelo caracter ",". Cada campo posterior será também separado por ",". Cada campo tem o formato <Nome>:<Tipo>[Tamanho], onde <Tipo> pode ser C para caracter, N para numérico, Y para (ano mês dia), M para (mês dia ano) e D (dia mês ano). Para o tipo C, caracter, o tamanho desejado pode acompanhar o campo. Por exemplo, no caso em questão:

- DESCRICAO:C500 é um campo de caracteres, de tamanho 500 e de nome Descrição.
- DATAFECHAMENTO:M é um campo de data, que segue o formato (mês dia ano) e tem o nome DATAFECHAMENTO, indicando a data de fechamento do incidente.

Os dados que foram inseridos no banco de dados foram os relevantes para a classificação como Descrição, Classificação, Severidade, Classe de Falha, Sintoma, Tipo de Componente da falha, Componente da Falha e Servidor. Além disso, foram inseridos a chave primária (Número OS), os dados ODC classificados manualmente na fase 1 ou automaticamente na fase 2 (Impacto ODC, Gatilho ODC, Componente Gatilho ODC) e, finalmente, se o *bug* pode ser reproduzido ou não (*bug* determinístico e não determinístico).

A **carga do banco de dados** (fase 4 da Figura 23) consistiu em fazer o *upload* do arquivo de dados utilizando o JMYSTIQ (JMYSTIQ, 2013). Esta fase do experimento automático corresponde a construção do Data Warehouse (Figura 6). Na Figura 25, mostra-se uma configuração de importação com o arquivo de meta dados e o arquivo de dados na ferramenta JMYSTIQ.

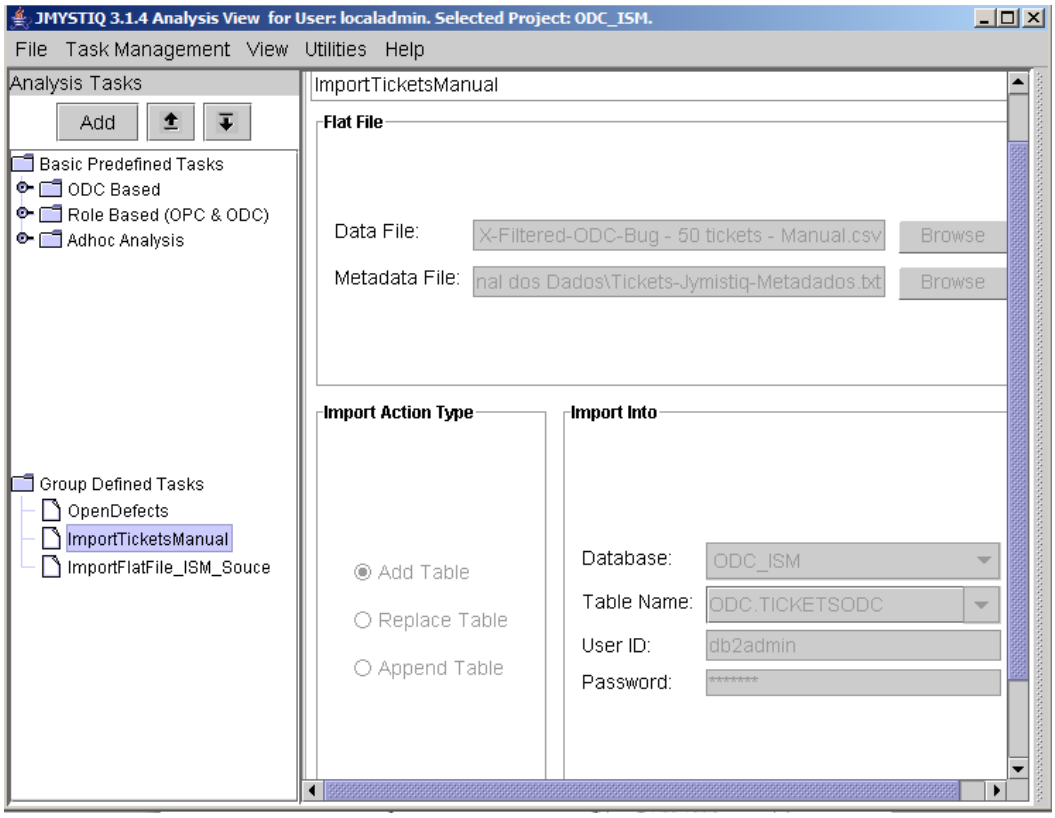

Figura 25 - Arquivo de configuração de importação na ferramenta JMYSTIQ.

Na configuração de importação da ferramenta JMYSTIQ define-se o arquivo de metadados, o arquivo de dados e o banco de destino, incluindo o nome da tabela, o usuário e a senha. A comunicação com o banco de dados de destino é feita por JDBC (*Java Database Connectivity*). Foram feitos alguns tratamentos de dados com relação à data de fechamento, que não tinha o campo segundos no formato hh:mm:ss e um tratamento para o separador do caracter “,”.

Ainda na fase 4, o campo Número OS foi importante como chave primária para a importação para o banco de dados e, consequentemente, para o modelo de dados.

O modelo de dados preparado para a próxima fase (o suporte a decisões) consiste das tabelas Descricao e Solucao, apresentadas na Figura 26. As tabelas de descrição e solução, “Descricao” e “Solucao” respectivamente, são populadas automaticamente com comandos SQL a partir da tabela principal TicketsODC utilizando triggers de bancos de dados (ver Figura 26).

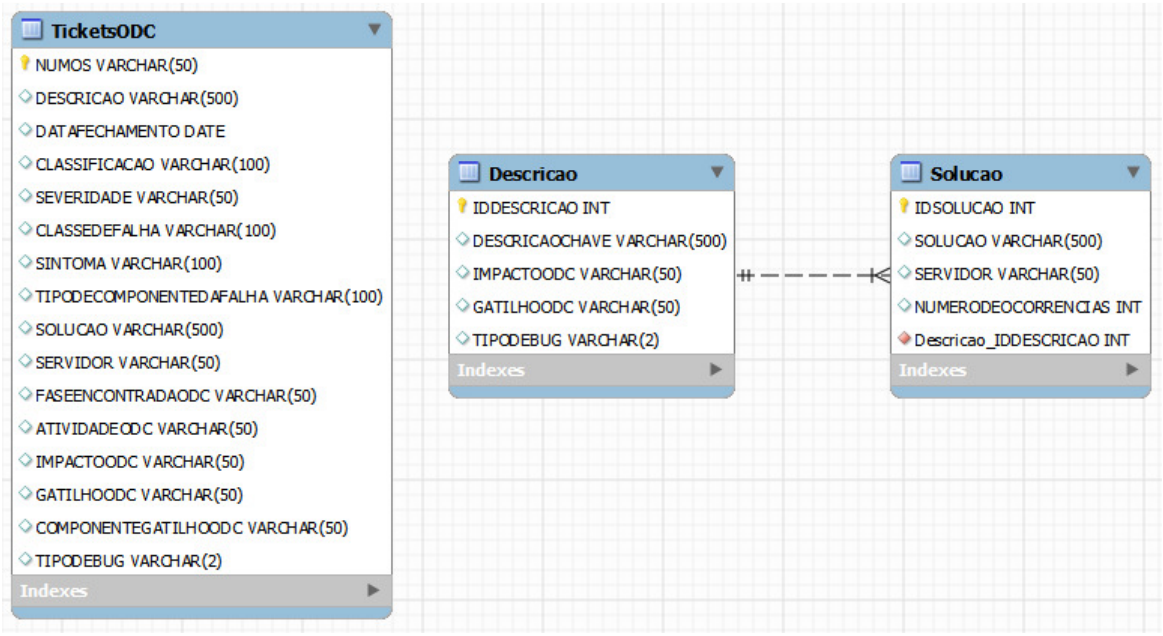

Figura 26 - Modelo de dados consistindo da tabela original junto com as tabelas (Descricao e Solucao) para as melhoras soluções.

A decisão das melhores soluções e direcionamento de suporte (fase 5 da Figura 23), consiste em propor, a partir dos dados nativos de uma ordem de serviço, qual o nível de suporte mais apropriado ao problema ou o time mais apropriado dentro de um mesmo nível (Banco de dados, Middleware, entre outros) para resolver o problema ou orientar uma solução mais provável para o problema. Esta fase corresponde no processo proposto a fase de obter as melhores soluções para o tipo de problema descrito (Figura 6).

Os dados inseridos no banco de dados permitem com que. as melhores soluções sejam ranqueadas a partir de comandos SQL para a montagem final do experimento. Pode-se observar na Figura 26, a tabela original (TicketsODC) carregada com os dados oriundos das fases 1 a 4.

Considerando o conjunto de tabelas da Figura 26, nesta fase do experimento foi concebido um artefato de consulta. O artefato faz uma consulta ao banco de dados e apresenta as melhores soluções (Figura 27).

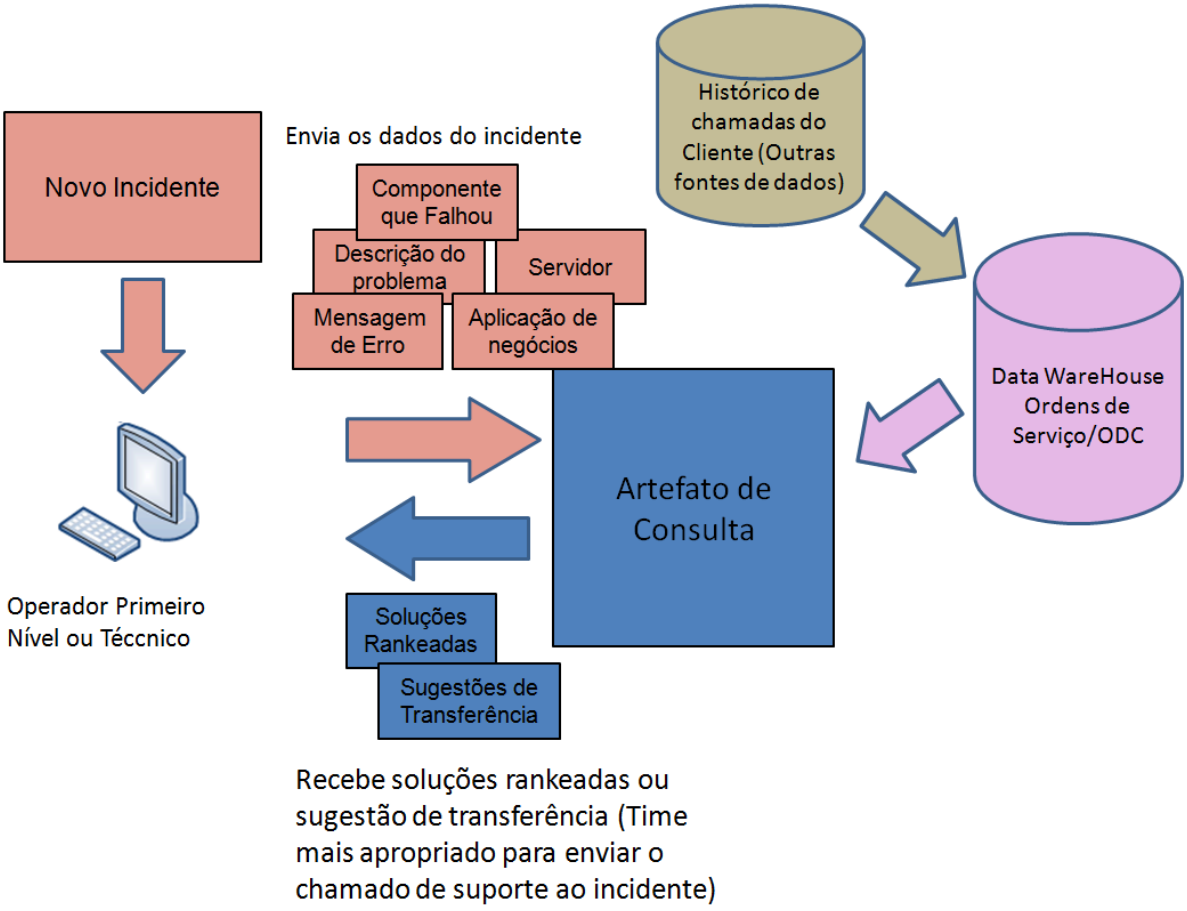

Figura 27 - Artefato de consulta para as melhores soluções.

Quando um novo incidente é computado, um operador técnico pode fazer uma consulta das melhores soluções a partir de palavras chave. As palavras chave podem ser a descrição do problema vinda do cliente, uma mensagem de erro específica, a localização do servidor, a aplicação de negócios impactada ou ainda um componente de software da falha (DB2, Websphere, etc). Apresenta-se um resultado com as soluções possíveis que estajam no data warehouse ao usuário e a solução desejada pode ser consultada. O mapeamento das palavras chave para os atributos do banco de dados é apresentado no Quadro 3:

Palavra Chave	Atributo no Data Warehouse
Descrição do problema	Descrição da Ordem de Serviço
Mensagem de Erro	Detalhamento (Drill-down) nível 3 ODC fonte
Aplicação de negócio impactada	Detalhamento (Drill-down) nível 2 do ODC Impacto (Aplicação de negócio ou Servidor) ou

	detalhamento ODC fonte Nível 2 (Localização do Erro se for o código da aplicação)
Componente da falha	Detalhamento (Drill-down) nível 2 do ODC gatilho

Quadro 3 - Mapeamento das palavras chave de pesquisa com o atributos de dados no *data warehouse*

Esta abstração apresentada no Quadro 3 é importante para diminuir a complexidade para a utilização do procedimento automático. As palavras chave foram definidas através de experiência no domínio de conhecimento do assunto e na informação mais comum. Descrição do problema é um dado que é bastante comum. A mensagem de erro aparece em muitas ocorrências. A aplicação de negócio é muitas vezes descritas pelo cliente quando ocorre uma indisponibilidade e o componente de software da falha é um dado muito usado no meio técnico. Estas palavras chave podem ser combinadas para se encontrar uma solução mais específica. A saída é formada pelas soluções ranqueadas por número de ocorrências, ou seja, a solução que foi mais vezes utilizada aparece em primeiro lugar, a segunda solução mais utilizada aparece em segundo lugar e assim por diante.

Outro resultado que aparece nesta fase, é se o tipo de problema pode ser resolvido pelo Suporte de Primeiro Nível, ou se deve ir para o Suporte de Segundo Nível ou de Terceiro Nível, indicando qual time pode resolver o incidente. Quando um novo incidente é aberto, o Suporte de Primeiro Nível (*Help Desk*) acessa o conjunto de artefatos aqui descrito e, por meio da descrição do problema e das outras palavras chaves descritas nesta seção, recebe a sugestão de transferir o incidente para o time mais apropriado. Desta forma, espera-se diminuir o número de transferências necessárias para resolver um incidente, diminuindo o tempo de atendimento ao cliente.

6 RESULTADOS

O número de transferências realizadas entre os times de suporte, que é encontrado na base de dados de controle do cliente, é um fator chave para medir a eficácia do método. Cada registro de incidente se inicia com um suporte básico e, a medida que uma solução não é encontrada, o incidente é repassado de um time especializado para outro mais especializado e de custo mais elevado para a Empresa "A". Adicionalmente, há um atraso para que esses times (recursos humanos) sejam engajados em um novo incidente e possam trabalhar nele. Em função da carga de trabalho dos times mais especializados, pode-se desperdiçar tempo aguardando que esses times estejam disponíveis e possam ser engajados em um novo incidente. Por outro lado, a qualidade nas transferências entre os times, que neste caso significa a precisão para enviar o problema ao time certo, é um outro fator relevante na redução do tempo de atendimento do incidente.

6.1 *Experimentos manuais*

Aplicando-se todo o processo proposto, foram geradas 50 amostras de incidentes para o mês de Abril de 2013 e 50 amostras de incidentes para o mês de Outubro de 2013. Os problemas de suporte foram referentes a mesma equipe (Middleware), e foram dos mais variados tipos, sistemas de arquivos, problemas de código da aplicação, CPU com alta utilização e etc. Uma amostra, relativamente a 10 incidentes do cliente "X", é apresentada no Quadro 4. As palavras chave usadas no novo processo foram a descrição do problema e a mensagem de erro.

Número do Incidente (Fictício)	**Número de Transferências Original**	**Número de Transferências com o Novo Processo**
1	49	44
2	5	5
3	16	13
4	79	74
5	14	11
6	30	2
7	63	62
8	78	77

9	24	4
10	19	7

Quadro 4 - Amostra relativa a 10 incidentes do cliente "X"

É possível verificar, analisando o Quadro 4, que há incidentes nos quais o Número de Transferências com o Novo Método é igual (como para o Incidente 2) ou mesmo pouco inferior (como para o Incidente 8) ao Número de Transferências Original. Porém, há também grandes diminuições no número de transferências, como nos Incidentes 6 e 9.

Para o Incidente 9 foi feito uma comparação de transferências entre os times (Figura 28). Neste aspecto pode-se observar que com o processo atual ocorreu uma intensa transferências entre os times de aplicação e o time de Nível 2. Isto aconteceu devido ao fato de um procedimento específico não ser conhecido pelo suporte Nível 2. Depois de algum tempo tentando obter o procedimento, o Nível 3 foi consultado e respondeu ao Nível 2. Com o novo método o procedimento estaria consolidado e disponível ao Nível 2, precisando somente de algumas transferências de validação e execução entre o time de aplicação e o segundo nível para resolver e fechar o incidente.

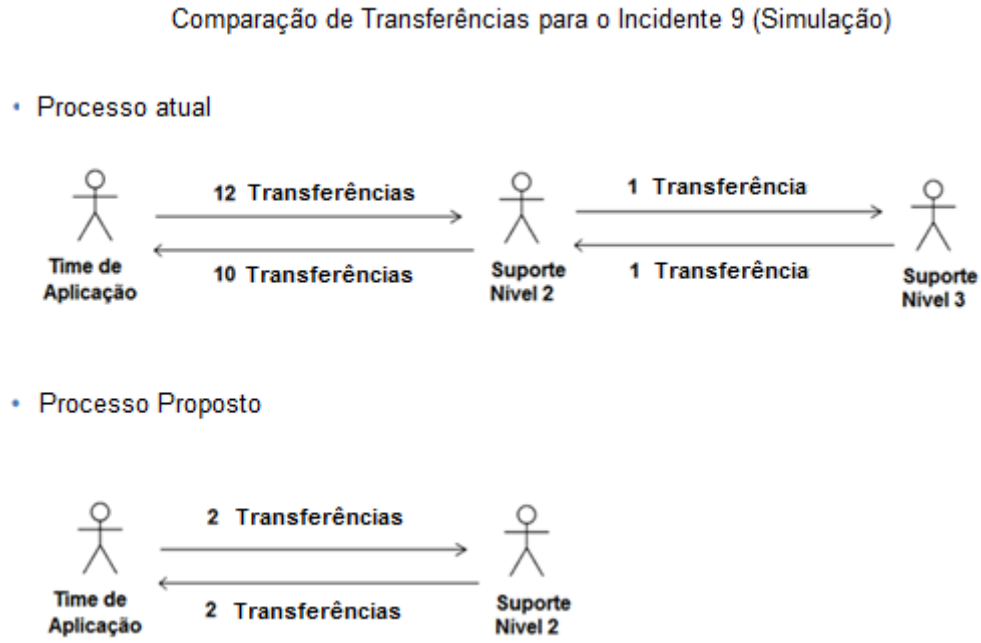

Figura 28 – Comparação de Transferências para o incidente 9 do cliente "X" do o processo atual com o processo proposto dentro (Simulação).

A respeito do ganho de tempo, foi analisado uma amostra de um incidente específico para o cliente "X" (Figura 29) representado pelo chamado de número 17866934 aberto em

30/03/2013 as 15:54 e restaurado o serviço em 31/03/2013. A causa raiz do problema, ou seja, a correção permanente só foi concluída em 1/5/2013. Se o segundo nível de suporte verificasse os procedimentos básicos sobre reciclar um servidor para esta aplicação, utlizando o data warehouse gerado pelo processo proposto, vários passos poderiam ser evitados. Com o processo proposto, através de uma simulação em comparação com o processo atual, a inicialização do aplicativo aconteceria em 30 minutos no máximo, pois sendo um bug determinístico, facilmente reproduzível, a solução seria rápida e padronizada. A redução de tempo para esta amostra em uma análise de tempo seria uma redução de 10 horas e 10 minutos no atendimento. Simulações de atendimento foram realizadas para comprovar este ganho de tempo utilizando um operador técnico.

Chamado Nr	Tipo de Bug	Data Abertura	Data Fechamento	Nr. Transf.	Nr. Transf. com o modelo
17866934	D	30/3/2013 15:54	1/5/2013 13:07	77	10

Figura 29 - Resultados do experimento manual para chamado específico do cliente "X".

O experimento manual gerou, para o cliente "X", os resultados apresentados na Figura 30.

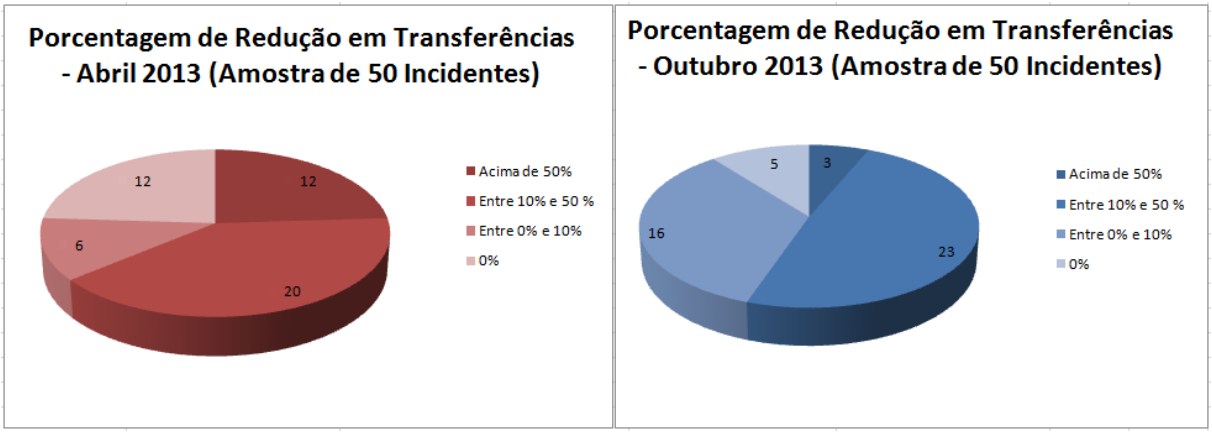

Figura 30 - Resultados do experimento manual para o cliente "X".

Como apresentado na Figura 30, em Abril de 2013, 20 registros de incidentes (40% da amostra) tiveram redução das transferências de 10 % a 50% com o método proposto neste trabalho. Em outubro de 2013, 23 registros de incidentes (49% da amostra) apresentaram também uma redução de 10% a 50% nas transferências.

Considerando todas as amostras para o cliente "X", e calculando a percentagem em que houve redução de tempo nos incidentes baseado no experimento, observa-se, na Figura 31, que houve redução de tempo nos incidentes em 92,50% dos incidentes.

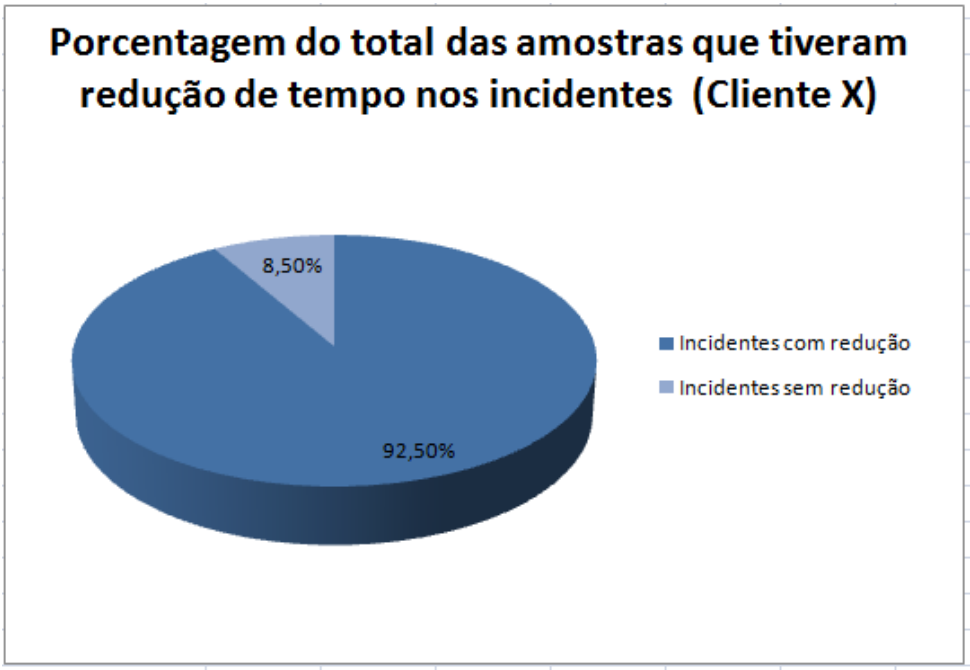

Figura 31 - Percentagem final de redução considerando o total de todas as amostras para o Cliente X.

Além disso, para 100% das ordens de serviço, o processo de classificação de *bugs* determinísticos (Bohr *bugs*) e não determinísticos (Mandel *bugs*) funcionou como esperado. Resumidamente, esse processo envolve o mapeamento do Gatilho ODC, do tipo de *bug* (determinístico ou não determinístico) e a análise dos registros do cliente para verificar se o problema poderia ser resolvido pelo nível mais básico (Bohr *bugs*) ou pelo nível mais especializado (Mandel *bugs*). Pesquisando nos registros do cliente, referentes às amostras das ordens de serviço, e através do número de chamado do cliente (*External Ticket No*), verificou-se que todos os Mandel *bugs* foram consertados pelo nível mais especializado. Por outro lado, vários Bohr *bugs* foram também resolvidos pelo nível mais especializado, sem que houvesse qualquer necessidade para isso, o que aumentou o custo de resolução de vários incidentes. A análise desses registros do cliente confirmou, dessa forma, que o processo de

classificação poderia indicar a resolução dos *bugs* determinísticos para o nível de suporte mais baixo.

Com relação ao cliente "Y", novamente foi aplicando todo o método descrito nas seções de 3.2 a 3.10.1, sendo geradas 50 amostras de incidentes para o mês de Abril de 2013 e 50 amostras de incidentes para o mês de Outubro de 2013. A Figura 32 apresenta os resultados de forma resumida.

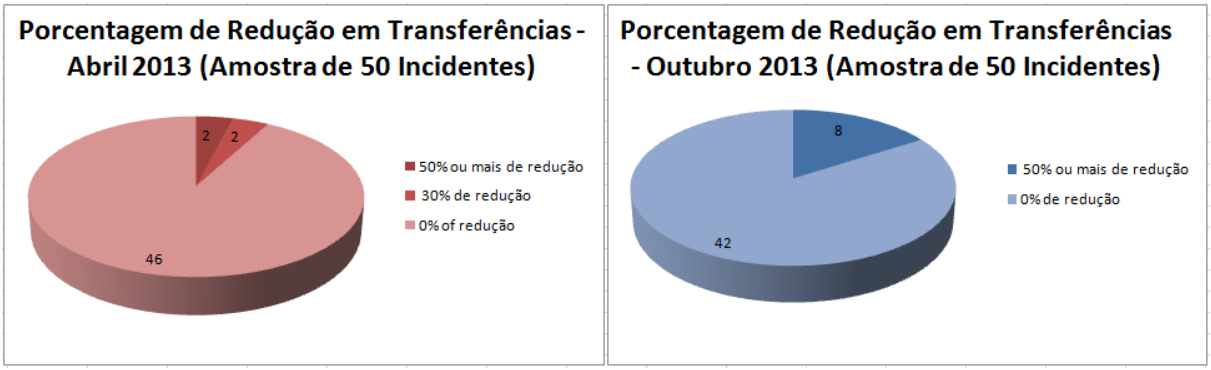

Figura 32 - Resultados do experimento manual para o cliente "Y".

Os resultados mostraram uma redução menor no número de transferências para a empresa cliente "Y", quando comparada como a redução para a empresa cliente "X". Entretanto, 71% dos incidentes, considerando as 2 amostras de abril e outubro de 2013, tiveram redução de tempo, conforme apresentada na Figura 33.

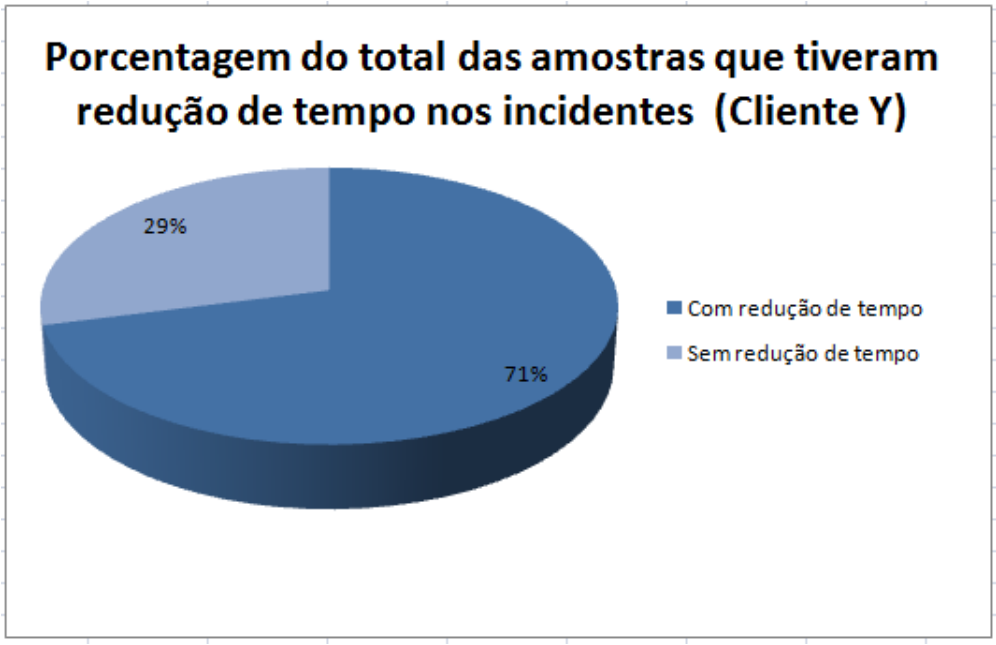

Figura 33 - Percentagem final de redução considerando o total de todas as amostras para o cliente "Y".

Para entender este resultado, foi feito um experimento adicional, a fim de verificar se os incidentes não resolvidos com o a atuação do time correto apareciam novamente para a mesma aplicação, ou para os mesmos servidores envolvidos. Isso ocorre, por exemplo, quando um *bug* não determinístico (*Mandel bug*) aparece em um incidente e um suporte de nível menos especializado realiza somente um Para/Inicia (*Stop/Start*), não passando o incidente para um nível mais especializado para resolver efetivamente o incidente, ou seja, contornando o problema ao invés de resolvê-lo. Um outro exemplo seria quando o suporte menos especializado restaura o incidente e não passa o mesmo para o nível mais especializado, a fim de iniciar um processo de investigação mais específico (gerenciamento do problema ou *Problem Management*). Para estes exemplos, fazendo uma investigação dos incidentes futuros, foi detectado que após algum tempo, outros incidentes aconteceram novamente relativos as mesmas aplicações de negócios ou a um mesmo servidor ou grupo de servidores. A Figura 34 exemplifica esta situação onde foi análisada Ordens de Serviço repetidas no provedor de serviço.

Incident	Summary	Priority	Status	Target Finish
IN4460537	Application A web is down.	1	CLOSED	10/25/13 6:07 AM
IN4496347	Application A in AG/WW/PDM and AG FMS Web not working	1	CLOSED	11/4/13 12:30 PM
IN4548784	Application A/ Servers B, C, D Issue :Need servers to be recycled	1	CLOSED	11/19/13 12:33 AM
IN4549071	Application A/ Servers B, C, D Issue :Need servers to be recycled	1	CLOSED	11/19/13 2:06 AM
IN4559844	Application A/ Servers B, C, D Issue :Need servers to be recycled	1	CLOSED	11/20/13 6:46 PM
IN4729632	Need server to be recycled for Application A - IGA application	3	CLOSED	1/15/14 5:35 PM
IN4767326	Application A - Backup File system is full	3	CLOSED	1/27/14 6:46 AM
IN4780210	Application A / Servers B,C, D URL: n/a Issue : Need servers recycle	1	CLOSED	1/23/14 5:51 PM
IN4847752	Ticket# :IN4847752 Reporter: Customer K (Email) Application: Application A -	1	CLOSED	2/12/14 6:44 PM
IN4955794	Application A/ Servers B, C, D Issue :Need servers to be recycled	1	CLOSED	3/15/14 1:30 PM
IN5037931	Query about MQ client on Application A	2	CLOSED	4/12/14 8:00 PM
IN5076848	Application A/ Servers B, C, D Issue :Need servers to be recycled	1	CLOSED	4/23/14 9:02 PM
IN5177026	Application A/ Servers B, C, D Issue :Need servers to be recycled	1	CLOSED	5/23/14 8:47 PM

Figura 34 – Incidentes repetidos para a Aplicação "A" e Servidores B, C e D.

Pode-se observar na Figura 34, que os incidentes que envolveram a aplicação "A" (Application A na Figura 34) e os servidores B, C e D (Servers B, C, D na Figura 34) se repetiram por quase 6 meses, mais precisamente de 19 de novembro de 2013 a 23 de maio de 2014 (verificar as linhas destacadas em verde na Figura 34). Isso significa que, se na ocasião do primeiro incidente ocorrido em 19 de Novembro de 2013, o incidente fosse restaurado pelo nível mais especializado, ou se fosse feita uma investigação mais aprofundada pelo nível mais

especializado, o incidente não se repetiria nos 6 meses subsequentes, o que gerou custos para o provedor de serviços.

A partir deste resultado, foi feito um estudo em todas as outras 99 ordens de serviço do cliente "Y", sendo confirmada a repetição do mesmo fenômeno em 71% delas. Em 29% o problema não se repetia, chegando-se ao ganho de tempo indireto conforme apresentado na Figura 33.

6.1.1 Explicação geral das Amostras Completas

Foram escolhidas uma amostras completas para os Cliente X e Y para demonstrar o trabalho manual realizado para classificar e agrupar soluções. Estas amostras são relativas ao período de Abril de 2013. Observando o Quadro 5, verifica-se os seguintes campos em comum:

Descrição da OS	Severidade do Chamado	Sintoma da Falha	Causa da Falha	Componente da Falha	Resolução	ODC Impacto/Severidade	ODC Impacto SVC	ODC Gatilho	ODC Gatilho SVC	ODC Fonte/Idade	ODC Fonte SVC	ODC Tipo de Defeito	Bohr/Mandel	Número de Redirecionamentos

Quadro 5 – Cabeçalho das Amostras completa para o mês de Abril-2013 para os cliente "X"e "Y"considerando os campos em comum

- Descrição da OS : A descrição do chamado junto ao cliente ou a monitoração automática.
- Severidade do Chamado: Sendo severidade 1 impacto severo a aplicação de negócios, Severidade 2 impacto parcial a aplicação de negócios.
- Sintoma da Falha: Seria uma denominação padrão ao sintoma observado no momento da Falha.
- Causa da Falha: Em geral seria uma descrição padronizada da causa da falha e que é preenchido geralmente quando se fecha a ordem de serviço, mas é um campo opcional e que muitas vezes pode ser ambíguo em relação ao que de fato aconteceu.

- Componente da Falha: Seria um campo que descreve com mais detalhas a falha e muitas vezes pode contrariar o que foi detectado no sintoma da Falha.
- Resolução: Informa suscintamente e muitas vezes de forma incompleta o que foi feito para resolver o problema. Este campo seria um dos motivos para a busca da base de dados do cliente. Na base de dados do cliente o campo de resolução é na maioria dos casos uma informação valiosa para complementar o campo resolução da base de dados do provedor de serviço.
- ODC Impacto/Severidade: Este campo é construído por meio de transformação de dados em data warehouse (HAN, KAMBER e PEI ,2012) para o impacto ODC junto ao cliente, sendo um campo mapeado a partir dos campos da ordem de serviço (severidade do chamado, a descrição da ordem de serviço (OS) e o sintoma da falha).
- ODC Impacto SVC: O Impacto SVC é concebido por um mecanismo de transformação de dados em data warehouse junto com operação OLAP de drill-down (HAN, KAMBER e PEI ,2012) de forma que é colocado como um detalhamento do atributo original ODC Impacto/Severidade.
- ODC Gatilho: Este atributo é concebido na transformação de dados a partir da Descrição do Chamado, Sintoma da Falha e Componente da Falha.
- ODC Gatilho SVC: Detalhamento do ODC Gatilho este atributo provém dos mesmos campos de origem do ODC Gatilho. O Detalhamento OLAP é realizado a partir dos mesmos campos do ODC gatilho mas observando por outra perspectiva que seria o componente de serviço envolvido no gatilho.
- ODC Fonte/Idade:O atributo ODC fonte/idade se refere quando o defeito é consertado, sendo criado por transformação de dados dos campos componente da falha, a causa da falha e a resolução.
- ODC Fonte SVC: Este atributo é um detalhamento do ODC Fonte/Idade, construído a partir dos mesmos campos originais que construíram o ODC Fonte/Idade so que visto da perspectiva do componente de serviço.
- ODC Tipo de Defeito: Definindo o defeito a ser consertado e padronizando junto ao modelo ODC, temos este atributo mapeado a partir do campo original Resolução.

- Classificação Bohr-Mandel: Este atributo é definido a partir da informação do ODC Gatilho anteriormente mapeado. Seria considerado uma operação OLAP obtida com os conceitos de Chillarege (2011).
- Número de Transferências: Este campo foi computado com a análise dos incidentes na base do cliente para verificar quantas transferências entre os times foi realizada.
- Com o Novo Processo: Neste campo que coloca-se o resultado da simulação quando se aplica o novo processo junto aos incidentes. A partir desta simulação obtem-se o primeiro indicador se o processo é eficiente em comparação com os registros obtidos pelo processo atual de atendimento aos incidentes.

6.1.2 Apresentação de uma Amostra Completa para o Cliente X

Descrição da OS	Severidade do Chamado	Sintoma da Falha	Causa da Falha	Componente da Falha	Resolução	ODC Impacto/Severidade	ODC Impacto SVC	ODC Gatilho	ODC Gatilho SVC	ODC Fonte/Idade	ODC Fonte SVC	ODC Tipo de Defeito	Tipo de Bug (Bohr/Mandel)	Número de Redirecionamentos	Com o Novo Processo	Justificativa
Unable to Login into ABACUS Application	2	CLIENT APP ISSUE	CLIENT APP ISSUE		Executed Failover on DB instance and recycled websphere. SDM Team coordinated with GLSSAXP_IBM_GL and middleware team and restarted the JVM's .Team was then able to connect to application.	Reliability	Abacus	Workload/Stress	Application	Developed In-House	Abacus	Algorithm/ Method	N	56	10	If were directed the problem to DB and after that recycles done with correct ID several steps would be saved
Intermittent Impact in CIW	2	CONFIGURATION		IHS permissions	IHS Recycled. RCA for permanent resolution	Capability	N/A	Software Configuration	Security infra	OutSourced	IHS permissions	Shipped Files	D	83	6	Recycle was not done imediatelty because sys admin didn´t have the procedures. RCA took long time due to few knowldge

LVPMA739:LVPMA739NODEIAMBATCHSERVER:WASBASE_GCTIMEHIG	2			App Issue	get the app team aproval to recycle the JVMs	Capability	ambatch	Workload/Stress	Application	Developed In-House	ambatch	Algorithm/ Method	N	7	7	This is a repeatable issue from ticket 17870890. There is another duplicated problem 17872067.
Atwork Enhanced Reporting Travel intermittently impacted.	2	APPLICATION OTHER	CONNECTIVITY	JVM when did stopped	provided information as requested	Requirements	Atwork Enhanced Reporting Travel	Software Configuration	Middleware	OutSourced	WebSpher e	Maintenance/Fix Dependencies	D	19	15	The JVM was stopped since a long time. it could be faced by low support level
Restart of JVM on ndm-mb01	2	APPLICATION OTHER	CONNECTIVITY	serverstatus not working	requested to kill the process and start jvm	Reliability	guniter	Workload/Stress	Middleware	OutSourced	WebSpher e	Process Conformance	N	11	11	No gain would be observed because The ticket was correctly sent to right level when not possible to check health of servers
LVPMA531::LNXBASE_HIGHCPUUSAGE	2				Problem caused by amex customer code related to log4J calls	Performance	GIDM	Workload/Stress	Application	Developed In-House	GIDM	Algorithm/ Method	N	49	47	This type of problem could be handle directly to high level. PMR was opened.
LVPMA531:WSFPP1LVPMA531GIDMSERVER:WASBASE_GCTIMEHIGH	2	APPLICATION ALERT		HGH CPU	Closed as duplicated of IMR 17871994.	Performance	GIDM	Workload/Stress	Application	Developed In-House	GIDM	Algorithm/ Method	N	13	11	This type of problem could be handle directly to high level. Data gathering could be get from high level

CARD INFO WEB	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	Requested RCA for system slowness	as per IBM labs, we did not see any issue in the logs. That is correct. This IMR is consequence of permission problems described on 17886985. So it can be merged with that same IMR and BCRS tkt. JVMs recycled	Reliability	Card Info Web	Software Configuration	Middleware	OutSourced	IHS permissions	Shipped Files	D	42	42	No gain would be observed because the high level was onlu engaged to do RCA
LVPMA738:LVPMA738NODEIAMBATCHSERVER:WASBASE_GCTIMEHIG	2	APPLICATION ALERT		Close as duplicated	Closed as duplicated	Performance	ambatch	Software Configuration	Application	OutSourced	ambatch	Process Conformance	N	5	1	This type of problem could be handle by low level As this problem is duplicated from incident nr 17839360
FX international payment	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	App Issue	transfer the ticket back requesting the servicrestore to follow up the RCA	Reliability	FX international payment	Software Configuration	Application	Developed In-House	FX international payment	Algorithm/ Method	N	34	25	No gain would be granted by framework as the problem was needed to be identified by all teams in a bridge. The main reason of reassigners was due to lack of process knowledge.
Cadence Application Servers are stopped (lvpwa535/lvpwa536)	2	APPLICATION SERVICE DOWN	IBM SW ERROR	App ibmasyncrsp was attempted to stop	Transferred ticket to L2 asking them to transfer to AIX team to investigate any command that could be issued.	Reliability	Cadence	Software Configuration	Middleware	OutSourced	WebSpher e	Process Conformance	D	104	98	Framework could helps to decrease if it has completed instructions to find who stopped application server
unable to Start jvm on server lvpma602	2	APPLICATION HANGING		JVM nost starting	Provided analysis of hung threads delaying JVM startup. JVM was restarted	Capability	cards	Startup/Restart	Application	Developed In-House	cards	Algorithm/ Method	D	19	19	Framework could not helps to decrease because problem was not correct identified to be passed directly to

																higher level
HO - LVPMA570::LNXBASE_LOWVIRTUALSTORAGE	2			App Issue	the app team was misslead to think that is a FIP problem. After identifying the Cause and researching known facts with the swap area usage, we implemented a possible fix by increasing the swap area/space to the maximum level compatible to server's memory parameters. This configuration setting was recommended to server services team and only after getting their confirmation that there won't be any application outage and performance degradation	Capability	ADE application	Software Configuration	Memory	OutSourced	swap area	Maintenance/Fix Dependencies	D	86	73	If Framework would be used a previous instruction on ticket 17779529 had the same problem and same instructions. Other reassigned was related to QA env.
HO - LVPMA532::LNXBASE_HIGHCPUUSAGE	2	APPLICATION ALERT	CLIENT SW ERROR	User requesting information about RCA.	Transferred to customer's queue. Downstream of the CMR #08921313	Performance	gidm	Workload/Stress	Application	Developed In-House	gidm	Maintenance/Fix Dependencies	N	72	49	Framework could helps to directly direct to higher level. A lot of effort was done with second level without any results
LVPMA739:LVPMA739NODEIAMBATCHSERVER:WASBASE_GCTIMEHIG	2	APPLICATION ALERT		Garbage collector	It was suggested to increase the heap size	Performance	AMBATCH	Workload/Stress	Memory	OutSourced	heap size	Maintenance/Fix Dependencies	N	6	5	Framework could help only to direct to higher level
LPPWA1313::LNXBASE_HIGHCPUUSAGE	2			Hight CPU	sugest to recycle the app server	Performance	upldcntr	Workload/Stress	Application	Developed In-House	upldcntr	Algorithm/ Method	N	11	5	Framework could helps to directly direct to higher level. A lot of effort was done with second level without

																any results
ASSURE Control Points are not functioning	2	APPLICATION OTHER	CONNECTIVITY	JVM stopped, alert needed	requested to contact Tivoli team. SDM Team coordinated with Networl support team and GLSSAXP_IBM_GL team to check the functioning of load balancer . Also requested to delete some old files and folders and this circumvented the issue .	Requirements	abacus	Software Configuration	Monitoring	OutSourced	WebSpher e	Maintenance/Fix Dependencies	D	77	10	If level 2 check basic procedures such as server stopped looking on framework regarding basic procedures. Several transferred steps could be avoided
Please restart GFG jvm	2	APPLICATION SERVICE DOWN	CLIENT SW ERROR		The new directories were created and IHS alogn with applications was restarted. Thanks	Reliability	GFG	Software Configuration	Middleware	OutSourced	IHS	Maintenance/Fix Dependencies	D	18	8	If checkList were observed on framework instructions the problem could be solved early
LVPMA532::LNXBASE_HIGHCPUUSAGE	2				Server's resources were not compromised at all	Capability	gidm	Workload/Stress	Application	Developed In-House	gidm	Maintenance/Fix Dependencies	N	19	9	Several Steps could be saved if done by higher level. Also this ticket was duplicated from IMR 17875434
LVPMA739:LVPMA739NODEIAMBATCHSERVER:WASBASE_GCTIMEHIG	2	APPLICATION ALERT		GC	It was suggested to increase the heap size. Duplicate of IMR #17839360	Performance	iambatch	Software Configuration	Application	OutSourced	WebSpher e	Maintenance/Fix Dependencies	N	18	18	No gain would be observed because The ticket was corretly sent to right level
Application logs not getting archived on lppwa1058/1063	2	CONFIGURATION	CONFIGURATION		Server Support team worked with Middleware and observed some issues with FileSystems, which got resolved. Upon confirmation, Application Support team validated the archival process.	Capability	cpiadapngaopenngaeaolngaeacqngaapimapply	Software Configuration	Middleware	OutSourced	OS	Maintenance/Fix Dependencies	D	61	61	No gain would be observed because The ticket was corretly sent to right level . The higher level took a long time to work on it and RCA took

																a long time to finish due to dificult to restore older files for analysis
HO - Username password not working for E2	2	APPLICATION ALERT	CLIENT SW ERROR	User requesting help with console user/passwd information.	Provided information.	Integrity/Security	GDT	Software Configuration	Security infra	OutSourced	webspher e	Maintenance/Fix Dependencies	D	6	2	Framwork wih correct information to get user/pwd to restart could help to make faster the solution
Intermittent Dips in OMS applications	1	APPLICATION OTHER	CLIENT SW ERROR		changes has been implemented to switch the GC policy to the GENCON policy, improving the performance of the application avoiding the memory leaks	Performance	OMS	Workload/Stress	MIddleware	OutSourced	webspher e	Maintenance/Fix Dependencies	N	17	13	If Framework would be used, the problem could be passed to higher level and avoid pager to a incorrect queue.
Please recylce LPPWA1125 Server	1	APPLICATION HANGING	CONTENTION	application had issues, query was taking long	code needs to be fixed	Reliability	ccsg	**Coverage**	Application	Developed In-House	ccsg	Algorithm/ Method	N	73	57	If consult framework, the decision should be go to Higher level, saving kill -3 and kill -9 procedures without specialized support
LVPMA531::LNXBASE_HIGHCPUUSAGE	2					Performance	gdim	Workload/Stress	Application	Developed In-House	gdim	Algorithm/ Method	N	45	37	If Framework would be used, the problem could be passed to higher level .
LVPMA739:LVPMA739NODEIAMBATCHSERVER:WASBASE_N	2				The issue will be fixed after moving the batch JVM to independent server and heap space will be increased to 8 GB. It will happen in 10 days	Performance	iambatch	Workload/Stress	Application	OutSourced	Memory/ Other Hardware		N	7	5	It could be closed before worked. It is a recurrent issue from ticket 17876279 using

OHEAP SPA																framework
LVPMA738:LVPMA738NODEIAMBATCHSERVER:WASBASE_GCTIMEHIG	2	APPLICATION SERVICE DOWN	CLIENT SW ERROR	server	Deplicated Problem	Performance	iambatch	Workload/Stress	Application	OutSourced	Memory/ Other Hardware		N	6	4	It could be closed before worked. It is a recurrent issue from ticket 17876279 using framework
LVPMA738:LVPMA738NODEIAMBATCHSERVER:WASBASE_NOHEAPSPA	2				Deplicated Problem	Performance	iambatch	Workload/Stress	Application	OutSourced	Memory/ Other Hardware		N	7	5	It could be closed before worked. It is a recurrent issue from ticket 17876279 using framework
LVPMA739:LVPMA739NODEIAMBATCHSERVER:WASBASE_GCTIMEHIG	2	CONFIGURATION	CONFIGURATION		Deplicated Problem	Performance	iambatch	Workload/Stress	Application	OutSourced	Memory/ Other Hardware		N	6	4	It could be closed before worked. It is a recurrent issue from ticket 17876279 using framework
The FX Ops back office is slow. Clients can not login.	1	APPLICATION OTHER	CLIENT SW ERROR		Provided analysis of hung threads and suggestion of traces.	Reliability	FX International Payments	Workload/Stress	Application	Developed In-House	FX International Payments		N	18	8	It could be closed before worked. It is a recurrent issue from ticket 17883015 using framework

Please restart the CV jvm	2	APPLICATION SERVICE DOWN	CLIENT SW ERROR	server	Contract viewer JVM is going OOM as the heap size insufficint to run the jobs. Working with middle ware to enhance the heap size RCADescription:Contract viewer JVM is going OOM as the heap size insufficint to run the jobs. Working with middle ware to enhance the heap size	Reliability	cvpw	Workload/Stress	Application	Developed In-House	cvpw		D	21	21	No Gain could be granted by framework because application staff team requested actions on ticket creation
Please recycle Beq portal jvms	2	NO PROBLEM	NO PROBLEM		Found no issues with syncNode. Beq portal servers get re-cycled successfully after upgradation of lvpma539/540. RCADescription:Beq portal servers get re-cycled successfully after upgradation of lvpma539/540.	Reliability	beq	Software Configuration	Middleware	OutSourced	WebSpher e	Maintenance/Fix |Dependencies	D	26	26	No Gain could be granted by framework because application staff team requested actions on ticket creation
Mime Type for image extension png is returned wrong	2	CONFIGURATION	CONFIGURATION		Configration setting information data for production server. RCADescription:Configration setting information data for production server.	Reliability		Software Configuration	Middleware	OutSourced	IHS	Maintenance/Fix |Dependencies	N	19	18	If Framework would be used, the problem could be passed to higher level .
LVPMA532::LNXBASE_HIGHCPUUSAGE	2	APPLICATION SERVICE DOWN	CLIENT SW ERROR	server	Downstream of the CMR #08921313	Performance	GIDM	Workload/Stress	Application	Developed In-House	GIDM	Algorithm/ Method	N	72	56	If framework would be used, the problem could be passed to higher level and high level be prepared to understand request
Please chck the Health of servers	2	APPLICATION ALERT	CLIENT SW ERROR	User reporting error 500 when accessing url.	Checked logs and SiteMinder errors detected.	Reliability	lvpma532	Software Configuration	Middleware	OutSourced	Siteminder	Maintenance/Fix |Dependencies	D	54	2	It could be soluted faster using framework. It is a recorred problem as it is a site minder

																problem that causes error 500 in web server
Card Info Web CIW unable to access in chennai	1	APPLICATION ALERT	CLIENT SW ERROR	Card Info Web CIW unable to access in chennai	Permissions for files under conf were corrected. This issue is related to permission problems when try to load site minder configuration files. The entries for favicon can disregard. The internal server error 500 is consequence of not loading site minder files.	Reliability	Card Info Web	Software Configuration	Middleware	OutSourced	Siteminder		D	37	2	It could be soluted faster using framework. It is a recorred problem as it is a site minder problem that causes error 500 in web server
Issue With LVPMA532 IHS - Webserver	2	CONFIGURATION		siteminder issues	ongoing	Capability	Card Info Web	Software Configuration	Middleware	OutSourced	Siteminder		D	13	2	It could be closed before worked. It is a recurrent issue from ticket 17886985 using framework
ASSURE Control Points are not functioning	2	APPLICATION OTHER		RCA	SDM Team coordinated with Networl support team and GLSSAXP_IBM_GL team to check the functioning of load balancer . Also requested to delete some old files and folders and this circumvented the issue .	Reliability	Abacus	Software Configuration	Middleware	OutSourced	OS		N	48	2	It could be closed before worked. It is a recurrent issue from ticket 17849573 using framework
LPPWA775::LNXBASE_LOWVIRTUALSTORAGE	2					Reliability	LPPWA775	Software Configuration	Storage	OutSourced	Virtual Storage		D	10	10	Framework could not helps because even if it was a false alarm it should be verified
MMDB: LPPWA924 recycle - JVM	2					Capability	MMDB	Workload/Stress	Application	Developed In-House	MMDB		D	40	35	Framework could not helps because it was a defined procedure from application team to be executed

																based on fact that application has funcionality problems. If framework used, some steps could be saved
IHS in lppwa1213 was not running	2	APPLICATION SERVICE DOWN	APPLICATION SERVICE DOWN	IHS	Advised to L2 how to proceed with IHS restart. It was not running fine due to site minder	Capability	lppwa1213	Workload/Stress	Middleware	OutSourced	Siteminder		D	92	79	It could be soluted faster using framework. It is a recorred problem as it is a site minder problem
LVPMA532::LNXBASE_HIGHCPUUSAGE	2	APPLICATION OTHER	CONNECTIVITY	high cpu	requested to ru linperf.sh. restrict trace logging in a change	Performance	LVPMA532	Workload/Stress	Middleware	OutSourced	WebSphere		N	24	21	If framework would be used, the problem could be passed to higher level
LVPMA928.GSO.AEXP.COM:DCSV8021WWAS6COREGROUPNODEAG	2	APPLICATION ALERT		Transport buffer size	no actions taken, alert raised during node federation	Performance	LVPMA928	Software Configuration	Middleware	OutSourced	WebSphere		D	10	9	Higher level would not needed to be engaged if framework would be used
LVPMA929.GSO.AEXP.COM:DCSV8021WWAS6COREGROUPNODEAG	2	CONFIGURATION		Transport buffer size	Alert was raised during node federation, no actions taken	Performance	LVPMA929	Software Configuration	Middleware	OutSourced	WebSphere		D	10	9	Higher level would not needed to be engaged if framework would be used
FXIP outage	2	APPLICATION OTHER	IBM SW ERROR	IHS	IHS Recycled. Permissions on plugin were changed	Reliability	FXIP	Software Configuration	Middleware/OS	OutSourced	IHS/Permissions		D	30	30	Correct procedures were done on early stages of incident, it means regenerating plugin and recycle IHS. Framrwork could helps in new issues

																related
XNET application is clocking	2	CONFIGURATION		WebSphere	Increase memory and CPU	Reliability	Xnet	Software Configuration	OS	OutSourced	swap area		D	43	31	Several steps could be saved using framework instructions for checking current stuck process and send to high level to verify CPU starvation logs
NODERCCOSERVERWAS_GCTIMEHIGH	22			WebSphere		Performance	RCCO	Workload/Stress	Application	Developed In-House	RCCO		N	13	10	If framework would be used, the problem could be passed to higher level
JVM wsfpp11vpma512aactacc Server not responding	2			WebSphere	recycling the server resolved the issue RCADescription:Server was out of memory	Reliability	actacc	Workload/Stress	Application	Developed In-House	actacc		N	9	8	Framework would not reduce so much the number of reassigned but it could helps low level to take java cores and heapdumps before recycle application and make easier root cause analysis work
LVPMA635:/LOGS:LNXBASE_LOWPERCFSAVAIL	2			OS		Capability	LVPMA635	Software Configuration	OS	OutSourced	Space dir		N	76	10	Framework could helps to drive the issue without engage higher level to see same result as low level. The solution if Franmework consulted, several steps could be taken and with

																/amex/cwme/bin/wasArchive.ksh procedure could be solved definitelly
LVPMA535:WSFPP1LVPMA533AACTACCSERVER having hung threads	2				recycling the server circumvented the issue RCADescription:Server was out of memory. Log 4J config can be the root cause of OOM	Capability	actacc	Workload/Stress	Application	Developed In-House	actacc		N	9	9	Framework could help to provide solution faster but times reassigned could not help

6.1.3 Apresentação de uma Amostra Completa para o Cliente Y

Descrição da OS	Severidade do Chamado	Sintoma da Falha	Causa da Falha	Componente da Falha	Resolução	ODC Impacto/Severidade	ODC Impacto SVC	ODC Gatilho	ODC Gatilho SVC	ODC Fonte/Idade	ODC Fonte SVC	ODC Tipo de Defeito	Tipo de Bug (Bohr/Mandel)	Número de Redirecionamentos	Com o Novo Processo	Time que proveu uma solução	Team que deveria resolver	Houve tempo Reduzido ?
PEW Server/URL: "Internal WAS Cluster: g01eiwascl005.ahe.pok.ibm.com g01ei	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	App Issue - g01eiwas012;g01eiwas013;g01eiwas014 - Product Entitlement Warehouse	The JVMs were recycled on the 3 servers (g01eiwas012; g01eiwas013; g01eiwas014)	Reliability	Pew	Workload/Stress	Application code	Developed In House	Pew	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
APReporting - can not access application	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	App Issue - g03zcxwas015.ahe.boulder.ibm.com - APReporting	The AppServer was recycled	Reliability	APReporting	Workload/Stress	Application code	Developed In House	APReporting	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Availability after Configuration Tool (ACT) Server/URL: https://www-304.ibm.com/servers/econfig/act	1	APPLICATION SERVICE DOWN		userID a3actadm seems to be locked.	CSI team recreated TSA keys	Capability	ACT	Software Configuration	Security	Outsourced	ATS Infra	Manage/Fix Dependencies	B	2	1	CSI	AHESS	Yes (Transfers)
Application: Routes-2-Financing Server/URL: g03ciwps043.ahe.boulder.ibm.com, g03ciwps044.ahe	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	App Issue - g03ciwps043;g03ciwps044 - Routes-2-Financing	The JVMs were recycled on both servers.	Reliability	Routes-2-Financing	Workload/Stress	Application code	Developed In House	Routes-2-Financing	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)

.boulder																		
Application/URL:Pes/Weborder and invoice server:g01aciwas027 Issue description:g01acxwas027Node.proc	1	APPLICATION SERVICE DOWN	IBM SW ERROR	Web Order and Invoice and Wpsci	Web team recycled WAS instance procurement_peswoiAS2.	Reliability	Pes/Web order	Workload/Stress	Application code	Developed In House	Pes/Web order	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Internal errors / page cannot be displayed when navigating/accessing portal	1	APPLICATION SERVICE DOWN	IBM SW ERROR	IBM Services.com	zLinux team fixed a typo in LDAP's config file. Web team refreshed a1ibmsvc.pwd file and recycled WAS instance.	Reliability	IBM Services.com	Software Configuration	Security	Outsourced	IBM Services.com	Shipped Files	B	1	1	AHESS	AHESS	Não
PBC-w3.ibm.com/hr/americas/pbc not accessible	1	APPLICATION OTHER	CONNECTIVITY	App	on server 013 we recycled the both JVMs	Reliability	PBC	Workload/Stress	Application code	Developed In House	PBC	Manage/Fix Dependencies	M	2	1	AHESS	CSI/Application	Yes (Transfers)
URL:http://w3.ibm.com/tools/expenses -500 SSL read timeout:	1	APPLICATION HANGING	CLIENT SW ERROR	URL:http://w3.ibm.com/tools/expenses -500 SSL read timeout:	Web team restarted the application WWER and the application is working fine.	Reliability	WWER	Workload/Stress	Java Asynch processes	Outsourced	Network	Manage/Fix Dependencies	M	1	1	AHESS	AHESS	Não
Separation and Leave of Absence - User is getting "Data access Error"	1	APPLICATION ALERT	CLIENT SW ERROR	recycle done	recycle done	Capability	Separation and Leave of Absence	Workload/Stress	Application code	Developed In House	Data Access	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Problem/issue: App is failing when processing batch job	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR		it was applied a new release to fix it.	Capability	MMAPS	Startup/Restart	Application code	Developed In House	batch job code	Algorithm/Method	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Reporter: zhenhuaw@cn.ibm.com Sev1 tkt (if you have): N/A Application/URL: g01aciwas066.ahe.pok.ibm.	1	APPLICATION ALERT	IBM SW ERROR	tools_wse_wmiAS4	Application was restarted according to customer request.	Reliability	tools_wse_wmiAS4	Workload/Stress	Server	Developed In House	g01aciwas066	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Application/URL:Supply Portal server:g01cihttp004.ahe.pok.ibm.com g01cihttp005.ahe.pok.ibm.com	1	APPLICATION SERVICE DOWN	IBM SW ERROR	Supply Portal	Web team refreshed pwd file from id a1waspes on g01cihttp004 and recycled IHS.	Reliability	Supply Portal	Software Configuration	IHS/Security	Outsourced	g01cihttp004	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	No

Oneteam. One of our jobs is not running due to a file that need to be removed	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	One of our jobs is not running due to a file that need to be removed	File was removed	Reliability	OneTeam	Software Configuration	ATS Job	Outsourced	./projects/w3/finance_access/web/server_root_w3/logs/oneteambatch/hrOffboardingTriggerProcessor.running	Install/Upgrade Dependencies	B	1	1	AHESS	AHESS	Não
Reporter: Dileep Mandapalli Application: IC2E Server/url: g01aciwpr006.ahe.pok.ibm.com Issue: Attach	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	IBM Cloud Commerce Engine (IC2E)	Add an additional group mapping so that the team processing Germany orders can access the attachments (implemented in ECR 4807435).	Capability	IC2E	Interaction	Application	Developed In House	File Servant Application	Manage/Fix Dependencies	M	1	1	SSIBMCOM	SSIBMCOM	Não
RESO Maximo AM - W3SR functinality not working for certain datas	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	W3SR functinality not working for certain datas	code fix reDeploy via ECR 4808218	Capability	RESO Maximo AM	Sequencing	Application	Developed In House	Application Code	Algorithm/Method	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Reporter: Cesar Moro fmoro@br.ibm.com Ticket#: Application: MMAPS Server/URL: https://g01aciwas002	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR		Code Fix implemented on other application version. ECR raised to Deploy it (4808218)	Capability	MMAPS	Startup/Restart	Application	Developed In House	Application Code	Algorithm/Method	M	3	2	AHESS	CSI/Application	Sim(Transferências)
GLTAMT - BH File processing failed.	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	App Issue - g01aciwas019.ahe.pok.ibm.com - GLTAMT	Blue harmony File processing bug was fixed by redeploying EAR - ECR# 4810109	Reliability	GLTAMT	Interaction	Application	Developed In House	Application Code	Algorithm/Method	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
lnxvm12/b01zedlog001.ahe.pok.ibm.com-High space used (90%) for /web	1	APPLICATION ALERT	CLIENT SW ERROR	lnxvm12/b01zedlog001.ahe.pok.ibm.com-High space used (90%) for /web	lnxvm12/b01zedlog001.ahe.pok.ibm.com-High space used (90%) for /web	Performance	Omnibus	Workload/Stress	File System	Outsourced	web FS Full	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	Não

Reporter: Yuan Hang Jiang/China/IBM Application: APReporting Server/URL: g03zcxwas015.ahe.boulder.ib	1	APPLICATION OTHER	IBM SW ERROR		Changed the ownership of /home/wassrvr and its subdirectories and was instalation files/directoires to wassrvr:users.	Reliability	APReporting	Software Configuration	File System	Outsourced	/home/wassrvr	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	No
GDMS Application didn't respond, retur of 500 error when users try access	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	WAS Server hanging out	Recycle of servers thru kill tasks	Reliability	GDMS	Workload/Stress	Application code	Developed In House	GDMS	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Separation and Leave of Absence - User is getting "Data access Error"	1	APPLICATION OTHER	CLIENT SW ERROR		Recycled hr_us_abouty ou_seploa JVM @ g03aeiwas009 and g03aeiwas010 servers.	Reliability	Separation and Leave of Absence	Interaction	Application code	Developed In House	Separation and Leave of Absence	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
ibm.com/training/us Issue: training search is down at this time. application recycle needed to r	1	APPLICATION OTHER	CLIENT SW ERROR	Application was hung	Application was recycled as per application team's request.	Capability	ITES Presentation Layer	Workload/Stress	Application code	Developed In House	ITES Presentation Layer	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Application: PWISV Server/URL:b01ciappha014.ahe.pok.ibm.com Issue: one of our batch programs are no	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR		one of our batch programs are not working...PurchaseEmail.ksh ; We recycled this process	Reliability	PWISV	Startup/ Restart	Application code	Developed In House	PWISV	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
IPPF - Application fail on key driver for PM action	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	Application Code bug - doesn't work keys	new code fix deployed by ECR#4821045	Reliability	IPPF	Interaction	Application	Developed In House	Application Code	Algorithm/Method	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Application: OneTeam Server/URL: b03acihttp003.ahe.boulder.ibm.com b03acihttp004.ahe.boulder.ibm.com	1	HANGING	HANGING	HTTP, FCGI	Checked if the process was up.	Capability	OneTeam	Software Configuration	IHS	Outsourced	httpd, Fcgi	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	Não
Server/URL: b03ciapp011.ahe.boulder.ibm.com Issue: File system full, we need a Web resource to clean	1	APPLICATION SERVICE DOWN	IBM SW ERROR	/projects/w3/finance_access/web/server_root_w3/logs	compressed old logs	Serviceability	OneTeam	Software Configuration	Application	Developed In House	Application Code	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	Não

g01aciwas033/034,g01aciwas034 Node.procurement_pescdaAS2 is	1	APPLICATION SERVICE DOWN	IBM SW ERROR	procurement_pescdaAS1 / procurement_pescdaAS2	Restart	Reliability	procurement	Workload/Stress	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
App:GARS Private Trading Exchange Error: 500 Internal Server Error	1	APPLICATION HANGING	CRASH		Fall back Change 4774205	Reliability	GARS	Software Configuration	Application	Developed In House	Application Code	Code Integration	B	2	2	Change Executor/Application (Operations)	Change Executor/Application (Operations)	Não
iRAM nodes are out of synch	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR		SEV-1 succeeded in synchronizing iRAM nodes	Reliability	IRAM	Workload/Stress	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
IA CRR-No available	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	Application Issue	server was hang we perform a recycle to release this hangs	Reliability	IA CRR	Workload/Stress	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
WSE-W - Server are hunging up status then we need AHE web team take recycle action asap	1	APPLICATION OTHER	IBM SW ERROR	there were no file spaces found for DB2	the issue was resolved by restarting the DB2 instance the logs were not being pushed because db2 was not able to push it to tsm .. and when the db2 instance was restarted it must have resolved the problem because of which now db scripts are able to	Reliability	WSE-W	Workload/Stress	Database	Outsourced	TSM client	Manage/Fix Dependencies	M	2	1	AHESS	DBA	Sim (Indiretamente)
b03edmq002.ahe.boulder.ibm.com- Cannot connect to the MQ server	1	APPLICATION OTHER	CLIENT SW ERROR	MQ server was unavailable	after MQ issue was fixed, middleware team recycled ATS and WAS java instances	Reliability	OneTeam	Software Configuration	MQ	Outsourced	ATS Infra	Manage/Fix Dependencies	B	2	1	AHESS	AHESS	Não
Server/URL: g01zcxwas019.ahe.pok.ibm.com Issue: dBlue has debug mode on last Sat. to collect logs fo	1	CONFIGURATION	CONFIGURATION	Log	Debug mode was turned off	Performance	dblue	Software Configuration	WebSphere	Outsourced	debug config	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	Não

Application: Brand Center Server/URL: g03acxwas016.ahe.boulder.ibm Issue: Application is down	1	APPLICATION OTHER	CLIENT SW ERROR	Lotus Connection server g25wascl46 was offline	Lotus Connection server g25wascl46 has been fixed Lotus connection team have no queue on TSRM, so I am closing the ticket for them.	Reliability	Brand Center	Software Configuration	Back end	Outsourced	Lotus Connection Server (g25wascl46)	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	Não
Application/URL: w3.ibm.com/services/ippf server: g03aciwas032.ahe.boulder.ibm.com Issue descripti	1	APPLICATION OTHER	CLIENT SW ERROR	Application was unavailable	Unknown, system recovered by itself	Reliability	IPPF	Workload/Stress	Application	Developed In House	Application Code	N/A	M	2	1	AHESS	CSI/Application	Sim (Indiretamente)
http://w3.ibm.com/hr/us/aboutyou/seploa/ Incident: User is getting "Data access Error"	1	APPLICATION OTHER	CLIENT SW ERROR	http://w3.ibm.com/hr/us/aboutyou/seploa/ Incident: User is getting "Data access	Web team recycled hr_us_abouty ou_seploaAS1 and hr_us_abouty ou_seploaAS2 application on g03aeiwas009 /010 servers.	Reliability	Separation and Leave of Absence	Interaction	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHE	CSI/Application	Sim (Indiretamente)
AHE/PTI: IEB is not working properly	1	APPLICATION SERVICE DOWN	IBM SW ERROR		app was recycled	Capability	PTI	Sequencing	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHE	CSI/Application	Sim (Indiretamente)
AHE: OSOL - Issue:OSOL application failed on "Run report" step.	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR		cron job was recycled	Capability	GDBM	Sequencing	Back end	Developed In House	Cron Jobs code	Manage/Fix Dependencies	M	1	1	AHE	CSI/Application	Sim (Indiretamente)
SalesOne S1SF/g03cxnp01056 - s1sf WebSphere is DOWN on g03cxnp01056	1	APPLICATION ALERT	CLIENT SW ERROR	SalesOne S1SF/g03cxnp01056 - s1sf WebSphere is DOWN on g03cxnp01056	SalesOne S1SF/g03cxnp01056 - s1sf WebSphere is DOWN on g03cxnp01056	Reliability	SalesOne S1SF	Workload/Stress	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHE	CSI/Application	Sim (Indiretamente)
AHE: ITES PL - getting error on some pages on production	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	Application Issue	recycle app server and ATS as request by AFP	Capability	ITESPL	Sequencing	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHE	CSI/Application	Sim (Indiretamente)
accessing the application url (https://w3-01.ibm.com/tools/prebill/gps.wss/billing/BAHome.ftl?link_i	1	APPLICATION HANGING	CLIENT SW ERROR	tools_gpbs_AS001 and tools_gpbs_AS002 application servers were hung	tools_gpbs_AS001 and tools_gpbs_AS002 application servers were recycled	Reliability	GPS	Workload/Stress	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHE	CSI/Application	Sim (Indiretamente)

Application: FIUI - https://w3-03.ibm.com/chq/finance/fiui/Server/url: b03acihttp004.ahe.boulder.ib	1	APPLICATION SERVICE DOWN	IBM SW ERROR	server unreachable	Recycled the server.Tried to kill processes but did not worked.Hence ,rebooted the server after taking backup and sys info.Hence,ticket was closed as server is up and running fine.	Performance	FIUI	Software Configuration	Middleware	Outsourced	IHS	Manage/Fix Dependencies	B	2	2	AHESS	AIX	Não
Mixed Address Database - Url not accessible .. Getting error 500	1	APPLICATION OTHER	CONNECTIVITY	App	JVMs were recycled;	Capability	MAD	Sequencing	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
Application: IBMLink Server: b03zcdrdb004.boulder.ibm.com and g03zcxrdb051.ahe.boulder.ibm.com g03zc	1	APPLICATION OTHER	CLIENT SW ERROR	Ricardo Souza Morais: We need to revert back the PWD of two users	I recycled the application as per application team's request	Reliability	IBM Link	Software Configuration	Security	Outsourced	Pwds	Manage/Fix Dependencies	B	1	1	AHESS	AHESS	Não
b03ciapp011.ahe.boulder.ibm.com - There are several stuck process that need to be killed	1	APPLICATION OTHER	CLIENT SW ERROR	Application was not working properly for one user.	ATS jobs were recycled and WAS Logs sent to Fabio Faria for investigation.	Capability	OneTEAM	Sequencing	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	2	1	AIX/ AHESS	CSI/Application	Sim (Indiretamente)
Business Agility Enablement Service	1	APPLICATION OTHER	CLIENT SW ERROR	FS from DB2 instance a1inps08 was mounted with errors	Zlinux team performed an ECR # 4839469	Capability	Business Agility Enablement Service	Startup/ Restart	Middleware	Outsourced	Database instance a1inps08	Manage/Fix Dependencies	M	2	2	AIX/ AHESS	DBA	Sim (Indiretamente)
g03eiwas054,g03eiwas055,g03eiwas056-Need servers to be recycled	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR	JVM name: finance_fmstmfAS	Restarted the JVM as requested	Reliability	TMF for FMS - IGA	Workload/Stress	Application	Developed In House	Application Code	Manage/Fix Dependencies	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
MMAPS/g01aciwas002 - Batch process is not running properly	1	APPLICATION SERVICE DOWN	CLIENT SW ERROR		a new EAR was implemented	Reliability	MMAPS	Startup/ Restart	Application	Developed In House	Application Code	Algorithm/Method	M	1	1	AHESS	CSI/Application	Sim (Indiretamente)
PROBLEM: GWA - Error - user called in reporting the outage on etotals gets the error "Internal Serv	1	APPLICATION SERVICE DOWN	IBM SW ERROR	application down - 500 error	After the failure, the channel could not re-establish the connection, causing issues when the Application tried to communicate from Green	Reliability	GWA	Software Configuration	Middleware	Outsourced	Web Sphere MQ	Manage/Fix Dependencies	B	3	2	NUSEF_HDSEV NUS_W_SSAHEMQ	NUS_W_SSAHEMQ	Sim(Transferências)

					Zone -> Blue Zone to Dedicated eTotals server. The connection was interrupted in a way that the receiver channel on queue manager B03CIMQ5 got hang, so the sender pair was not able to connect. I tried to recycle the receiver channel to get it working again, but the commands didn't take effect. The only alternative was to recycle the dedicated queue manager B03CIMQ5. After the recycle the application started working properly again. Validation was done during group chat.												

Quadro 6 - Amostra completa para o mês de Abril-2013 para o cliente "Y"

6.1.4 Caso Prático de Simulação de Transferências

Fazendo-se uma verificação em detalhes de um incidente do Cliente "X" visa-se exemplificar a eficiência de redução de tempo para o método proposto mesmo que em simulação. Na Figura 35, com o procedimento atual, verifica-se que as transferências ocorrem entre o suporte Nível 2 e o time de aplicação de forma desnecessária para executar os procedimentos de coleta de dados para um incidente de CPU em alta utilização.

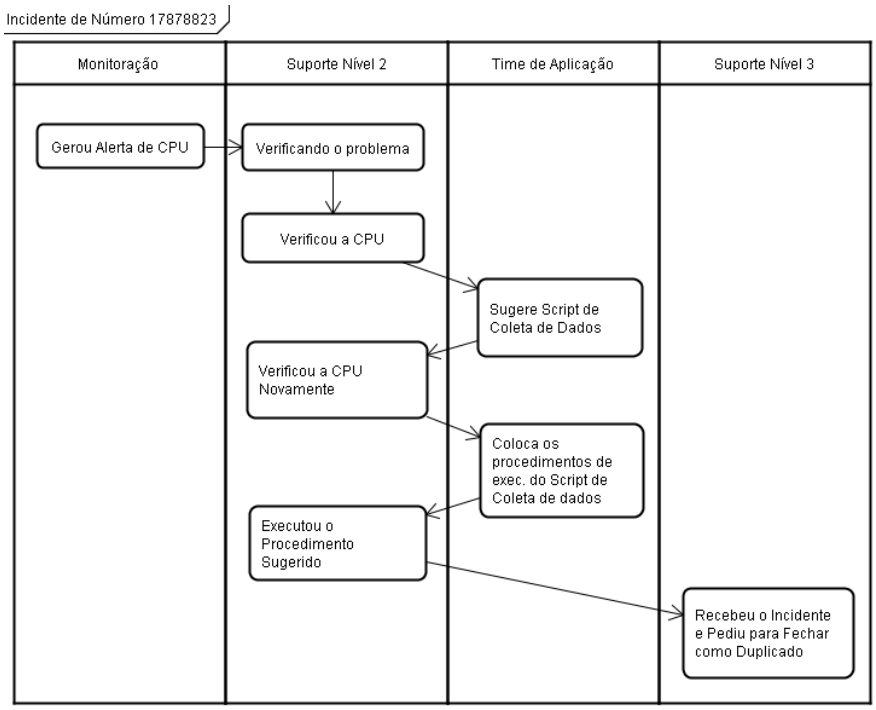

Figura 35 – Mapeamento Detalhado de Transferências de um Incidente do Cliente "X"

A partir do ponto que é feita a coleta dos dados pelo nível 2, depois de várias transferências, que é enviado o incidente para o nível 3 que pede para fechá-lo devido ao fato que tem um outro incidente anterior ainda aberto para a mesma aplicação, porém, os dados coletados são considerados valiosos para uma verificação em maior profundidade pelo Nível 3.

Com o método proposto, analisando a Figura 36, temos para o mesmo Incidente de alta utilização de CPU, mesmo considerando uma simulação, que o procedimento específico ja foi inserido anteriormente no data warehouse do processo proposto pelo domínio de experts. Verificando este procedimento no mesmo data warehouse, este pode-ser utilizado diretamente pelo suporte nível 2 e repassado ao nível 3 que pode utilizar esta informação para uma resolução prática.

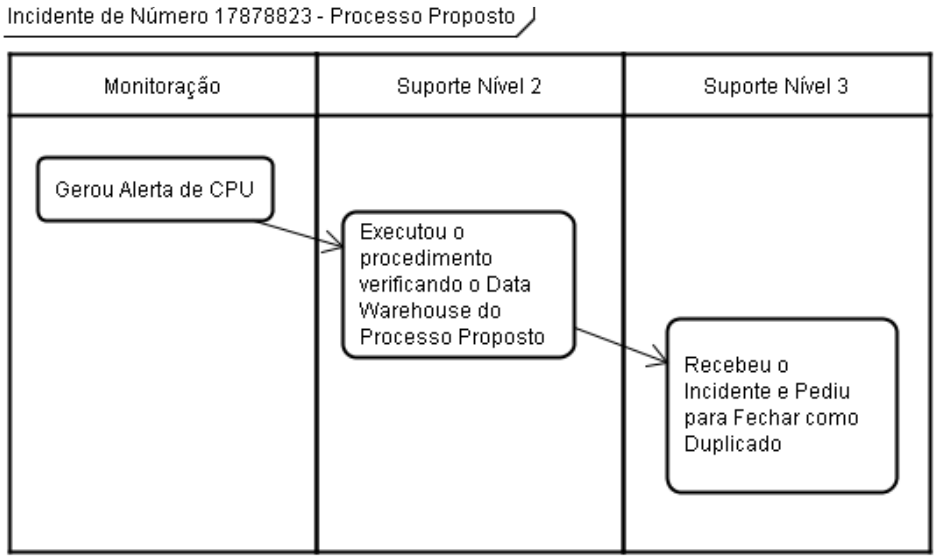

Figura 36 – Mapeamento Detalhado de Transferências com o Processo Proposto de um Incidente do Cliente "X"

6.2 *Experimento automatizado*

Detalhes de construção do experimento automatizado podem ser visualizados na seção 5.2. A respeito da classificação automática, os resultados mostram que o atributo tipo de *bug* (se determinístico ou não determinístico), alcançou um bom resultado, em torno de 90% de precisão utilizando algoritmos de mineração de dados (*data mining*) e tendo como pressuposto que o Gatilho ODC foi corretamente classificado. Outros atributos ODC alcançaram até 50% de precisão através de árvores de decisão, até 63,2% utilizando-se Naive Bayes e até 67,3% com Multilayer Perceptron (Rede Neural Artificial).

Sobre o *data warehouse*, consultas (*queries*) em linguagem SQL foram implementadas e testadas. Como exemplo, a seguir uma consulta para soluções a partir da palavra de pesquisa HIGHCPU:

```
Connect to ODC_ISM;
SELECT DESCRICAO, IMPACTODCSVC, IMPACTOODC, GATILHOODC,
TIPODEBUG, SOLUCAO
from ODC.TICKETSODC
WHERE ODC.TICKETSODC.DESCRIPTION LIKE '%HIGHCPU%'
```

Na Figura 37 pode se observar as soluções selecionadas a partir do comando SQL anterior. A solução com maior incidência foi a solução "instructions", com 2 ocorrências, o

que significa que instruções devem ser passadas para uma coleta de dados de High CPU. Esse procedimento é o correto para este tipo de situação. Os dados de coleta de CPU são enviados para o suporte do produto, com a finalidade de realizar uma análise mais aprofundada do problema, obtendo, por exemplo, *dumps* de memória do Java. Essas instruções são padrão e podem ser aplicadas para quase todas as máquinas virtuais Java, podendo ser colocadas como procedimentos de verificação inicial (*check list*) que podem ser realizadas por recursos humanos pouco especializados.

ODC.TICKETSODC [Filtered]

DESCRICAO [VARCHAR(500)]	SERVIDOR [VAR...	IMPACTOODC [V...	GATILHOODC [VARC...	TIPODEBUG [V...	SOLUCAO [VARCHAR(500)]
LVPMA531::LNXBASE_HIGHCPUUSAGE ...	LVPMA531	Performance	Workload/Stress	N	
HO - LVPMA532::LNXBASE_HIGHCPUUSAGE	LVPMA532;gidm	Performance	Workload/Stress	N	Transferred to customer...
LVPMA531::LNXBASE_HIGHCPUUSAGE		Performance	Workload/Stress	N	
LPPWA1313::LNXBASE_HIGHCPUUSAGE	lppwa1313	Performance	Workload/Stress	N	sugest to recycle the ap...
LVPMA532::LNXBASE_HIGHCPUUSAGE		Performance	Workload/Stress	N	
LVPMA531::LNXBASE_HIGHCPUUSAGE		Performance	Workload/Stress	N	instructions
LVPMA532::LNXBASE_HIGHCPUUSAGE	LVPMA532	Performance	Workload/Stress	N	
LVPMA531::LNXBASE_HIGHCPUUSAGE	LVPMA531	Performance	Workload/Stress	N	
LVPMA532::LNXBASE_HIGHCPUUSAGE		Performance	Workload/Stress	N	instructions

Figura 37 - Saída de soluções a partir do banco de dados para o select por palavra chave HIGHCPU

Também, a partir do comando SQL anterior, pode-se verificar, na Figura 31, que a solução “instructions” tem o tipo de *bug* (TIPODEBUG na Figura 37) não determinístico, o que implica em encaminhar o incidente para um Suporte de Segundo Nível (ver a Figura 2), o que elimina, imediatamente, uma das transferências que o método atualmente em uso pela Empresa “A” indicaria.

6.3 *Limitações nos testes*

Algumas situações não puderam ser testadas nas simulações, como a utilização em uma situação em tempo real para verificar a eficácia do processo. Após a unificação dos experimentos automátizados em uma aplicação única e a autorização da empresa de suporte, um teste real será possível.

7 DISCUSSÃO

A precisão de classificação dos incidentes em *bug* determinístico e não determinístico alcançou 100% com o experimento manual em ambos os clientes "X" e "Y". Este resultado de 100%, com o experimento manual, baseia-se no fato de que os tipos de bug dos incidentes foram corretamente classificados a partir de uma classificação correta do Gatilho ODC, porque há um mapeamento direto entre estas variáveis. No experimento automático depende totalmente da acurácia de acerto do ODC gatilho onde alcançou uma precisão em torno de 90%. Os outros atributos ODC alcançaram uma precisão de 50% a 67,3% com o experimento automatizado.

Houve resultados diferentes para o cliente "Y" em relação ao cliente "X", quando se analisaram as amostras após a aplicação do experimento manual. Uma das causas possíveis para isso pode estar relacionada ao fato de que o cliente "X" é totalmente externo à Empresa "A", com outra infra estrutura de atendimento, enquanto que o cliente "Y" faz parte da estrutura interna da Empresa "A", o que pode tornar a avaliação dos incidentes mais difícil em relação ao cliente "X" do que em relação ao cliente "Y".

A redução do número de transferências para o cliente "Y" foi de somente 8 a 16% dos incidentes. Por outro lado, para o mesmo cliente "Y", em diversas ordens de serviço, foi observado que, quando o Suporte de Primeiro Nível (ver a Figura 2) trabalhou no problema, o mesmo não encontrou a causa raíz, somente realizando uma reinicialização para restaurar o serviço assim que possível, não transferindo o problema para um nível mais especializado a fim de realizar uma investigação. Portanto, mesmo havendo o mesmo número de transferências com o processo atual utilizado pela Empresa "A" e com o método proposto neste trabalho, a qualidade nas transferências (para o time de suporte correto) com o processo proposto foi maior. A maior qualidade está aqui relacionada com a transferência para o time correto. Quando o incidente foi transferido para o Suporte de Primeiro Segundo Nível, quando deveria ter sido transferido para os Suportes de Terceiro Nível (ver a Figura 2), o problema não foi resolvido corretamente. Neste caso, o incidente somente foi apenas remediado, o que ocasionou a repetição do mesmo tipo de incidente por diversas vezes. Indiretamente, o método proposto diminuiria o tempo pois, quando o primeiro problema ocorresse, seria resolvido sem gerar mais novos incidentes no futuro. Isto foi observado por meio de experimentos de pesquisa com as mesmas aplicações envolvidas nas ordens de serviço das amostras.

Vários incidentes no cliente “X” mostram que o Suporte de Segundo Nível tentou resolver problemas direcionados para times mais especializados, tais como os *bugs* não determinísticos (Mandel *bugs*). No cliente “X”, que teve um número de transferências por incidente muito maior que o cliente “Y”.

No cliente “X”, comparando o processo atual com o processo proposto, mesmo que uma transferência somente fosse diminuída com o processo proposto, poderia-se economizar muito tempo, pois quando um chamado vai para uma fila de suporte, pode haver uma demora considerável, em função da carga de trabalho para as equipes, para que um recurso humano acesse o registro do incidente e comece a trabalhar. Além disso, dependendo do provedor de serviço, pode-se ter filas de suporte especializadas para cada produto de software, aumentando-se o tempo de recepção do incidente e atuação.

Analisando os *bugs* determinísticos (Bohr *bugs*) para o cliente “X”, observou-se que, em várias amostras, não havia uma descrição correta de resoluções de incidentes comuns, fazendo com que o Suporte de Primeiro Nível tivesse que transferir o incidente para o nível mais especializado. Se o primeiro nível consultasse na própria base do provedor pela aplicação “Z”, em muitos casos vieram procedimentos de resolução incompletos.

Utilizando-se o processo proposto, poderia-se definir procedimentos específicos voltados para cada aplicação de negócios. Esse tipo de característica é muito necessária, principalmente em provedores que possuem times distribuídos em diversos locais, de forma regional e internacional, e em provedores de computação em nuvem. Procedimentos pré-definidos poderiam ser colocados no *data warehouse* a partir da interação com experts no domínio do conhecimento, e estes poderiam ser executados pelo Suporte de Primeiro ou Segundo Níveis, evitando transferências desnecessárias para os níveis mais especializados de suporte.

Os níveis especializados de suporte, em algumas amostras de ordens de serviços, quando se analisa conjuntamente o histórico (*log*) de chamados do cliente, não encontraram facilmente a solução de vários incidentes que tinham uma ocorrência praticamente idêntica. Isto é consequência do conhecimento técnico não ser compartilhado entre turnos de trabalho diferentes de um mesmo time de suporte. As experiências de um recurso humano técnico são pouco compartilhadas entre os outros integrantes de time de suporte.

Em outras amostras, o nível especializado não notou que o problema era alinhado com o seu nível de conhecimento, transferindo novamente o problema para o nível menos

especializado, aumentando o tempo de resolução do incidente. Com o método proposto, esses problemas poderiam ser minimizados com um compartilhamento melhor do conhecimento e com a confirmação de que um determinado problema corresponde a um determinado nível de suporte.

O método proposto apresentou problemas relacionados a dados de entrada incompletos e muitas vezes em branco. Esses dados deveriam ser preenchidos pelos recursos humanos técnicos envolvidos nas ordens de serviço. Isto indica que a qualidade dos dados é fundamental para se alcançar bons resultados com o método proposto. Alguns campos em branco criaram dificuldades para a análise dos dados. A fase de limpeza dos dados foi importante para eliminar entradas que não agregariam nenhuma informação para o *data warehouse*. Esse problema poderia ser minimizado com um treinamento dos recursos humanos técnicos e uma contínua verificação da qualidade dos dados de entrada no sistema de ordens de serviço. Qualquer sistema de tomada de decisão se beneficiaria com essas ações.

Outro ponto importante a mencionar, é que os registros das ordens de serviço não foram suficientes em vários casos para suprir uma correta classificação, e que a base de dados de registros do cliente foi necessária para o método proposto ser utilizado. Outras bases de dados poderiam ser integradas para melhorar a eficácia do método proposto, como por exemplo, bases de conhecimento de produtos de softwares (Bancos de dados, Middlewares, Sistemas Operacionais, etc.).

O método proposto obteve uma redução de tempo em 71% dos incidentes para o cliente “Y”e 92,5% dos incidentes para o cliente “X”. Baseado nessas amostras, o método proposto foi mais rápido do que o método de gerenciamento de incidentes atual quando aplicado a estes mesmos clientes,

O experimento automatizado teve um desempenho mais rápido que o experimento manual. Considerando um mundo real, com uma empresa de suporte com alto volume de incidentes ocorrendo diariamente, em turnos de vinte horas, sete dias por semana (24 x 7), a solução automatizada seria a mais recomendada para implementar o método proposto.

Comparando como outros trabalhos como o AutoODC (HUANG et al. 2011), o processo proposto nesta pesquisa, possui uma parte de pré-processamento bem mais elaborada. No modelo AutoODC é aplicado o algoritmo de mineração de dados diretamente aos dados de texto puros sem uma estruturação prévia dos dados.

Em relação a pesquisa de Wang, Akella e Ramachandran (2010) onde temos sistema de análise de texto chamado Analisador e Recomendador de Requisições de Serviço ou Service Request Analyzer and Recommended (SRAR), que extrai informação crítica e pertinente a partir de documentos em texto de requisições de serviço de bases de dados de uma empresa de software usando técnicas de mineração, mas com uma inteligência de dados possui uma parte de pre-processamento bem mais completa com contextualização da palavra pesquisada utilizando o mecanismo de mala de palavras (*Bag of words*), porém não tem uma classificação de defeitos e funciona em um conjunto de problemas mais restrito (Serviço de redes). O processo proposto possui uma abordagem mais ampla por abordar problemas mais diversos.

8 CONCLUSÃO

O processo proposto foi suficiente para separar de forma clara os incidentes na abertura dos chamados e os problemas no fechamento dos chamados para uma montagem do data warehouse de suporte de incidentes com as soluções associadas a classe de problemas. Observou-se na base de dados uma clara separação de soluções.

Os problemas mais fáceis de identificar, os *bugs* determinísticos (Bohr *bugs*), podem ser resolvidos pelo Suporte de Primeiro Nível pois, sendo bem determinados e reproduzidos facilmente, esses *bugs* já possuem uma ação pré-determinada e que com o tempo pode se tornar mais elaborada, conforme o seu uso e eficácia. Problemas não determinísticos (Mandel *bugs*) podem ser associados a times com mais conhecimento.

A base de conhecimento utilizada nas soluções associadas a cada classe de incidentes, em muitos casos, não foi suficiente para diminuir o tempo de atendimento a problemas mais complexos, pois detalhes da resolução não foram bem especificados. A partir desta deficiência, são propostas duas ações: a primeira seria o preenchimento dos dados de resolução de forma clara pelos recursos humanos que estão atendendo o serviço, propondo um modelo de preenchimento dos dados da resolução de forma sucinta, mas objetiva; a segunda seria uma associação a outras fontes de dados, como as ferramentas de controle de chamados do próprio cliente. Esta segunda solução foi utilizada nos experimentos e resolveu em parte o problema de dados incompletos.

O processo proposto com o ODC e com as operações OLAP sobre o mesmo, que geraram novas extensões para o ODC, permitiu a otimização do processo de suporte, pois viabiliza a análise das transferências para os times corretos para a solução dos incidentes.

Com os resultados obtidos a partir das quatro amostras diferentes de dois clientes com características diferentes, o método proposto possibilita uma diminuição de tempo no atendimento dos incidentes e uma redução nos custos e esforço para resolvê-los. O modelo proposto indica uma ação direcionada para resolver os incidentes em menos tempo do que o método de processar incidentes e resolvê-los que a Empresa “A” utiliza atualmente.

A limitação deste trabalho consistiu em não realizar uma aplicação completa. Teve-se vários artefatos independentes, mas uma implementação final seria a melhor alternativa para explicar melhor o processo proposto.

8.1 *TRABALHOS FUTUROS*

A partir de uma classificação baseada no ODC e em ITIL, podem ser realizados trabalhos para melhorar a extração dos dados antes de classificá-los. Esses trabalhos podem incluir mineração de texto e comparações entre diversos algoritmos.

Ainda em relação à classificação, uma abordagem possível, e que não foi encontrada durante a produção deste trabalho, seria a proposição de uma unificação com o modelo OPC (*Orthogonal Problem Classification*). Esse modelo (IBM RESEARCH, 2013) coleta informações a partir das Chamadas de Serviços do Cliente (*Customer Service Calls*). Em resumo, são relatórios de problemas oriundos dos clientes, que poderão cooperar com o ODC.

Em relação ao método de determinação do problema, a clusterização das soluções semelhantes pode ser uma alternativa interessante.

Outras bases de conhecimento como, por exemplo, informações a respeito de produtos de suporte como banco de dados ou *middlewares* poderão ser associadas ao modelo proposto. Outras fontes de informações do cliente, como bancos de dados de chamados também podem ser associadas ao modelo.

Com relação ao *ranqueamento* das soluções, uma possibilidade seria a utilização de Redes Neurais Artificiais para auxiliar no processo de decisão para a escolha da melhor solução. As Redes Neurais Artificiais poderiam analisar o uso e a eficácia das diversas soluções para um problema, indicando a melhor solução para um problema.

Outro ponto importante seria a utilização de ontologias para melhorar o domínio de conhecimento (terminologia) e por consequência melhorar as soluções para a classe de problemas. Este ponto será uma base para uma melhoria do processo, juntamente com a criação de uma aplicação completa com software livre.

REFERÊNCIAS

AAMODT, A.; PLAZA, E.. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, vol. 7, no. 1, pp. 39-59, Mar 1994.

ASSUNÇÃO, Marcos D.; CAVALCANTI, Victor F.; GATTI, Maira A. de C.; NETTO, Marco A. S.; PINHAEZ, Claudio S.; SOUZA, Cleidson R. B. de. , "Scheduling with Preemption for Incident Management: When Interrupting Tasks is not Such a Bad Idea". Proceedings of the 2012 Winter Simulation Conference C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds., 2012.

ALEN, E.; SEAMAN, C.A. Likert Scales and Data Analyses. Quality Progress, vol. 40, no. 7, pp. 64-65, Jul. 2007.

BEIZER, B.; VINTER O., Bug taxonomy and statistics - Technical report. Software Engineering Mentor, pp. 2630, 2001

BISHOP, Christopher M. Pattern Recognition and Machine Learning, 1.ed. Reino Unido: Springer, 2006.

BUCKLEY M.; CHILLAREGE, Ram. Discovering relationships between service and customer satisfaction. Software Maintenance, 1995. Proceedings., International Conference on , EUA, vol., no., pp.192,201, 17-20 Oct 1995.

CHILLAREGE, Ram. Understanding Bohr-Mandel bugs through ODC Triggers and a case study with empirical estimations of their field proportion. Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on, EUA, pp. 7-13, 2011.

CHILLAREGE, Ram, Comparing four case studies on Bohr-Mandel characteristics using ODC, in *Software Reliability Engineering Workshops (ISSREW)*, 2013 IEEE Int. Symp. on, Passadena, CA, pp.285-289, 2013

CHILLAREGE, Ram., ODC - a 10x for Root Cause Analysis, in *Proc. RAM 2006 Workshop*, Berkeley, CA, pp 1-7, 2006.

CHILLAREGE, Ram., ODC for Process Measurement, Analysis and Control.Fourth International.Conference on Software Quality, 1994.

CHILLAREGE, Ram; BHANDARI, Inderpal S.; CHAAR Jarir K.; HALLIDAY, Michael J. ; MOEBUS, Diane S. ; RAY, BONNIE K.; WONG, Man-Yuen. Orthogonal defect classification-a concept for in-process measurements, IEEE Transactions on Software Engineering, EUA, Vol. 18, pp. 943-956, 1992.

COTRONEO, D.; Pietrantuono, R.; Russo, S.. Testing techniques selection based on ODC fault types and software metrics. Journal of Systems and Software, vol. 86, no. 6, pp. 1613-1637, Jun. 2013.

DIAO, Yixin, Staffing Optimization in Complex Service Delivery Systems. Network and Service Management (CNSM). 2011 7th International Conference on, EUA, pp. 1-9, 2011

EJARQUE, J.; DE PALOL, M.; GOIRI, I.; JULIA, F.; GUITART, J.; BADIA, R.M.; TORRES, J. SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers. IEEE Fourth International Conference on, Espanha, pp. 8-15, 2008.

GONDRA, I., Applying machine learning to software fault-proneness prediction, The Journal of Systems and Software, EUA, v. 81, pp. 186-195, 2008.

GRAY, J. Why do computers stop and what can be done about it?, Technical Report 85.7, PN87614, Tandem Computers, Cupertino, 1985.

GROTTKE, Michael; TRIVEDI, Kishor S. A Classification of Software Faults, in Proc. Sixteenth International IEEE Symposium on Software Reliability Engineering, EUA, pp. 1-2, 2005.

HAN, Jiawei; KAMBER, Micheline; PEI, Jian. Data Mining: Concepts and Techniques. 3. ed. Waltham: Morgan Kaufmann, Boston,EUA,2012.

HASSOUNA, A.; TAHVILDARI, L.. An effort prediction framework for software defect correction. Information and Software Technology, vol. 52, no. 2, pp. 197-209, Feb. 2010.

HE, Wang; HAO, Wang; ZHIQING, Lin. Improving Classification Efficiency of Orthogonal Defect Classification Via a Bayesian Network Approach. Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on, China, pp.1-4,2009.

HUANG, LiGuo; NG, Vicent.; PERSING, Isaac; GENG, Ruili; BAI, Xu; TIAN, Jeff. AutoODC: Automated Generation of Orthogonal Defect Classifications. Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on, EUA, pp. 412-415, 2011.

IBM DB2 Express-C. Disponível em: http://www-01.ibm.com/software/data/db2/express/download.html. Acesso em: 20 dez 2012, 15:51.

IBM RESEARCH Center for Software Engineering, Examples of the Use of ODC. Disponível em: <http://www.research.ibm.com/softeng/ODC/ODC.HTM>. Acesso em 10 fev. 2013, 23:35.

IEEE 829, Disponível em:

http://standards.ieee.org/findstds/standard/829-2008.html. Acesso em 3 mar. 2013, 19:32.

INMON, W.H., KELLEY, C. Rdb/VMS, Developing the Data Warehouse, QED Pub. Group, Boston, EUA, pp. 225, 1992.

INMON, W. H. Building the Data Warehouse, Wiley Publishing, Indianopolis, Indiana, 4ª Edição. 2012.

ITIL OFFICIAL SITE, Disponível em:

http://www.itil-officialsite.com/. Acesso em 7 mar. 2013 16:23.

JMYSTIQ, Disponível em: http://www.ibm.com/developerworks/rational/library/aug06/gu/. Acesso em 14 jun 2013 15:36.

KIM, Sunghun; ZIMMERMANN, Thomas; WHITEHEAD, E. James Jr.; ZELLER, Andreas. Predicting Faults from Cached History. Software Engineering. In: ICSE 2007, 29th International Conference on, EUA, pp. 489-498, 2007.

KIMBALL, R., Data Warehouse Toolkit, Makron Books do Brasil Editora, São Paulo, 1998.

KNUTH, D. E.. The errors of TEX. Jornal Software–Practice and Experience, vol. 19, Issue 7, pp. 607-685, Jul. 1989.

LEE, C.K.H., CHOY, K.L.; HO, G.T.S.; CHIN, K.S.; LAW, K.M.Y.; TSE, Y.K.. A hybrid OLAP-association rule mining based quality management system for extracting defect patterns in the garment industry. Expert Systems with Applications, vol. 40, no. 7, pp. 2435–2446, Jun. 2013.

MANHAES, M.; EMER, M. C.; BASTOS, L. C.. Classifying defects in software maintenance to support decisions using hierarchical ODC. In Computer on the Beach 2014, Florianópolis, SC, pp. 283-292,2014.

MUNSON, John C; NIKORAB, Allen P.; SHERIF, Joseph S. Software faults: A quantifiable definition. Advances in Engineering Software, EUA, vol. 37, no. 5, pp. 327-333, 2006.

OYETOYAN, T. D.; CRUZES, D. S. ; CONRAD R., A study of cyclic dependencies on defect profile of software components. Journal of Systems and Software, vol. 86, no. 12, pp. 3162-3182, Dec. 2013.

ORACLE MySQL WorkBench.Disponível em:

http://dev.mysql.com/downloads/workbench/. Acesso em: 20 dez 2012, 14:31

PARKA, Byoung-Jun; OHB, Sung-Kwun; PEDRYCZ, Witold. The design of polynomial function-based neural network predictors for detection of software defects. Information Sciences, Coréia do Sul, vol. 181 no. 2, pp. 257-378, 2011.

PASCHKE, Adrian; SCHNAPPINGER_GERULL, Elisabeth. A Categorization Scheme for SLA Metrics. Multi-Conference Information Systems (MKWI06), Passau, Alemanha, vol. 1, no. 5, 2006.

PECCHIA, A; COCOTRONEO, D. ; KALBARCZYK, Z. ; IYER, R.K.Improving Log-based Field Failure Data Analysis of multi-node computing systems. Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on, Itália, pp. 97-108, 2011.
PENTAHO. Disponível em: http://www.pentaho.com/. Acesso em: 03 Mar. 2013, 19:03

PLOSKI, J.; ROHR, M.; SCHWENKENBERG, P.; HASSELBRING, W.. Research issues in software fault categorization. ACM SIGSOFT Software. Engineering Notes, vol. 32, no. 6, Nov. 2007.

PODGURSKI, Andy; LEON, David ; FRANCIS, Patrick; MASRI, Wes; MINCH, Melinda; SUN, Jiayang; WANG, Bin. Automated Support for Classifying Software Failure Reports. ICSE '03 Proceedings of the 25th International Conference on Software Engineering, EUA, pp. 465-475, 2003.

REFORMAT, Marek ; IGBIDE, Efe, Isolation of Software Defects: Extracting Knowledge with Confidence. Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on, Canada, pp. 114-119, 2005.

SAHOO, Ramendra K.; SIVASUBRAMANIAN, A.; SQUILLANTE, Mark S.; ZHANG, Yanyong.Failure Data Analysis of a Large-Scale Heterogeneous Server Environment. Proceeding DSN '04 Proceedings of the 2004 International Conference on Dependable Systems and Networks, EUA, pp. 772-781, 2004.

SARAIYA, Nirav; LOHNER, Jason E.; BAIK, Jongmoon.Monitoring and Improving the Quality of ODC Data using the ODC Harmony Matrices: A Case Study".Proc. Fourth International Conference on Software Engineering Research, Management and Applications, EUA, pp. 407-411,2006.

SAUVE, J.; MOURA, A.; BARTOLINI, C.; TRASTOUR, D. A decision support tool to optimize scheduling of IT changes. Integrated Network Management, 2007.IM '07. 10th IFIP/IEEE International Symposium on, Brasil, pp. 343-352,2007.

SERRANO, N. Bugzilla, ITracker, and other bug trackers. IEEE Software, Espanha, vol. 22, no. 2, pp. 11-13, 2005.

SCHUGERL, P.; RILLING, J.; CHARLAND, P. Mining Bug Repositories - A Quality Assessment. Computational Intelligence for Modelling Control & Automation. 2008 International Conference on, Canadá, pp. 1105 - 1110, 2008

TIEJUN, Pan; LEINA, Zheng; CHENGBIN, Fang. Defect Tracing System Based on Orthogonal Defect Classification.Computer Science and Software Engineering, 2008 International Conference on ,China, vol. 2, pp. 574-577, 2008.

THUNG, Ferdian; LO, David; JIANG, Lingxiao. Automatic Defect Categorization. In: 19th Working Conference on Reverse Engineer ; Canadá, pp. 205-214, 2012.

UR-RAHMAN, N.; HARDING, J.A. Textual data mining for industrial knowledge management and text classification: A business oriented approach. Expert Systems with Applications, Reino Unido, vol. 39, no. 5, pp. 4729–4739, 2012

VOLPATO, Gilson L. Método Lógico para a Redação Científica. 1.ed. São Paulo: Editora Best Writing, 2011.

WANG, C.; AKELLA, R.; RAMACHANDRAN. S., Hierarchical service analytics for improving productivity in an enterprise service center. In: Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM '10), New York, NY, pp. 1209-1218, 2010.

WEKA. Disponível em:

http://www.cs.waikato.ac.nz/ml/weka/. Acesso em: 25 Ago 2013, 19:10

WITTEN, I. H.; FRAN, E.; HALL, M. A., Output: Knowledge Representation. In Data Mining: Practical Machine Learning Tools and Techniques, Third Edition, San Francisco: Morgan Kaufmann Publishers Inc, ch. 3, pp 72, ch4, pp 97,99, 2011.

XIAO J.; AFZAL, W.,. Search-based resource scheduling for bug fixing tasks. In: Proc. 2nd Int. Symp. Search Based Software Engineer (SSBSE), Benevento, Italy, pp. 133-142, 2010.

ZHANG, Huiji, A System for Fast Positioning SLA Violations. Wireless Communications, Networking and Mobile Computing, 2009.WiCom '09. 5th International Conference on, China, pp. 1-3, 2009

ZHANG, Ling-Feng ; SHANG, Zhao-Wei.Classifying feature description for software defect prediction.Wavelet Analysis and Pattern Recognition (ICWAPR), 2011 International Conference on, China, pp. 114-119, 2011.

APÊNDICE A - REVISÃO BIBLIOGRÁFICA

Para montar o embasamento teórico e o modelo proposto foi feita uma revisão Bibliográfica em relação ao assunto abordado. Foram pesquisados os portais *Science Direct* (http://www.sciencedirect.com), IEEE (http://ieeexplore.ieee.org) e ACM *Library* (http://dl.acm.org), utilizando-se as palavras chave descriminadas a seguir, no período entre 2002 e 2014. A partir dos portais citados, foram analisados os artigos de congressos e periódicos. As palavras chaves pesquisadas para a revisão sistemática foram as seguintes:

- Software Defect Prevention
- Software Defect Prediction
- KDD Software Defects
- KDD Incident Management
- Incident Management Restoration
- Time-Bounded Incident Management
- Software Defect Taxonomy
- Software Defect Prevention Classification
- Software Bug Tracker
- Defect Causal Analysis
- ODC
- Defect Rank

Tomando-se como exemplo a palavra chave *Software Defect Prevention*, para os portais pesquisados a chave de pesquisa ficou como apresentada no Quadro 7:

Site	String de Busca
Science Direct	Software AND Defect AND Prevention AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Software) AND Defect) AND Prevention) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014

ACM Library	(Software and Defect and Prevention) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 7 - Exemplo de chaves de Pesquisa que foi usada na revisão sistemática.

As demais chaves de pesquisa seguem o mesmo padrão, incluindo-se o conector *AND* para as palavras compostas, se necessário, e formatando de acordo com o portal de busca. Mais detalhes de todas as pesquisas realizadas estão no Apêndice A deste trabalho.

Do total de publicações obtidas de periódicos e conferências, a partir da pesquisa das palavras chave descritas anteriormente, e de artigos encontrados a partir de referências dos artigos inicialmente encontrados, foram selecionadas, a partir da leitura do título e do resumo, inicialmente, 108 publicações relevantes. Este conjunto inicial passou por uma segunda filtragem. Para as publicações científicas selecionadas, foi aplicada a Escala de Likert (ALEN; SEAMAN, 2007) para determinar a relevância ou importância da publicação em relação ao tema que abordado, lendo-se a introdução e conclusão dos artigos e atribuindo valores de -2 a +2 para cada publicação, onde cada valor tem um significado de relevância segundo o Quadro 8.

Valor	Likert	Significado ao projeto
-2	Discordo Totalmente	Nenhuma relevância
-1	Discordo Parcialmente	Pouca relevância
0	Não Concordo e nem Discordo	Sem posição formada
1	Concordo Parcialmente	Relevante
2	Concordo Totalmente	Muito Relevante

Quadro 8 - Quadro explicativo da Escala de Likert para os artigos selecionados

Com os valores da Escala de Likert atribuídos para cada publicação, definiu-se a revisão sistemática inicial, dando-se prioridade de pesquisa aos artigos com muita relevância, seguidos pelos relevantes. Os artigos de pouca relevância e nenhuma relevância foram descartados. Outro levantamento realizado, foi a distribuição das publicações selecionadas na primeira filtragem, ou seja, as 108 publicações, entre países que foram responsáveis pela elaboração, de forma a verificar a aderência do tema em estudo nos principais centros de

pesquisa. Para os casos de autores de diversos países, considerou-se a nacionalidade do primeiro autor. Também foram montados gráficos com o resultado da pesquisa para cada palavra chave, relacionando o número de artigos e o ano da publicação, a fim de verificar a evolução do interesse dos pesquisadores ao longo do tempo. Os resultados completos estão apresentados a seguir

Gerenciamento de Serviços de Manutenção de Software (Restauração de Incidentes e SLA)

Relativamente ao assunto Gerenciamento de Incidentes e o Acordo de Nível de Serviço ou SLA (*Service Level Agreement*) há uma grande ênfase nos custos de manutenção de software para se manter uma alta disponibilidade dos serviços e nas penalidades importas para a violação de SLA. Essas penalidades estão relacionadas a demora na restauração do serviço e a uma efetiva solução dos problemas para que eles não ocorram novamente como descrito em (ZHANG, 2009).

Existem estudos que direcionam para uma categorização do SLA de forma a se evitar erros de classificação e custos associados como descrito em (PASCHKE; SCHNAPPINGER_GERULL, 2006).

Outros trabalhos são direcionados para o estudo do controle de mudanças e como isto impacta o SLA. Soluções para esses problemas são providas, como a melhor alocação de recursos, a Análise de Pareto sobre as diversas violações de SLA, o Método de Árvore de Experiência na violação de SLA e outros que utilizam métodos de escalonamento otimizado de mudanças (SAUVE et al,, 2007).

Existem também estudos que tratam da virtualização de recursos, de modo que se consiga uma performance aceitável para se garantir o SLA acordado com o cliente, como em (EJARQUE et al., 2008).

Taxonomia e Classificação de Defeitos de Software, Incluindo o ODC

Sobre a classificação de defeitos foi encontrada em muitos trabalhos uma verificação da importância da classificação para facilitar a priorização dos defeitos para o negócio e para o diagnóstico das causas (PODGURSKI et al., 2003).

A dificuldade em classificar os defeitos, em relação ao número e ao tipo de defeitos, resulta em que a maioria das definições utilizadas levem a que não haja garantia de que dois indivíduos diferentes, quando olhando o mesmo conjunto de falhas e as mesmas definições de falhas façam uma contagem de mesmo valor. Com relação a este ponto, há trabalhos que se focam na classificação baseada na versão do software como em (MUNSON; NIKORAB; SHERIF, 2006).

Outros métodos de classificação de software para detecção de erros se baseiam no código fonte (código estático), através de estudos que provem uma evidência empírica que correlaciona algumas métricas e erros de software. Por usar essas métricas, o prognóstico de defeitos de software é usualmente visto como uma classificação binária, na qual os módulos de software são classificados em inclinados a falhas (fault-prone) e não inclinado a falhas (non-fault-prone), como descrito em (GONDRA, 2008).

Os métodos mais vezes encontrados nesta revisão foram IEEE 829 (IEEE 829, 2013) e o ODC (IBM RESEARCH, 2013).

O número de resultados de trabalhos utilizando o ODC foi muito maior e os mais promissores métodos de classificação de software, através do ODC, descrevem um processo automático chamado AutoODC, que utiliza um mecanismo de texto estruturado dos incidentes e aplica mineração de textos utilizando Maquinas de Suporte Vetorial (*Supporting Vector Machines*) (HUANG et al., 2011)

O *framework* autoODC apresentou as seguintes características e resultados (HUANG et al., 2011):

- O AutoODC usa um framework de anotação de relevância que habilita uma classificação baseado em um conjunto rico de dados de treinamento, possibilitando uma interação com experts no domínio de conhecimento para criar um classificador robusto.

- O classificador AutoODC alcançou a acurácia de 80,2% quando comparado a classificação manual, onde a acurácia da solução é computada como a percentagem de registros classificados corretamente pelo *framework*.

- Como um complemento da classificação manual, o AutoODC melhora o índice de confiança porque reduz o conjunto de investigação que um especialista tem que examinar para fazer a classificação.

Esse modelo é baseado nas seguintes fases, como explicado em (HUANG et al., 2011):

- Pré processamento de defeitos. Onde é gerado um campo de descrição textual do defeito e um campo textual sumário
- Aprendizado ODC. Neste ponto é abordado um classificador utilizando máquinas de suporte vetorial para o aprendizado do conjunto de treinamento.
- Classificação. A partir do conjunto de treinamento o classificador é aplicado. São colocadas as extensões, como as anotações adicionadas pelos experts que vão monitorar a acurácia e melhorar o rendimento do classificador.

Um grande potencial de melhoria deste modelo é principalmente na área de pré-processamento, onde, devido à heterogeneidade dos dados, poderia ser aplicado um processo de *Data Warehousing* para eliminar os dados desnecessáios e para de agrupar outras fontes de dados, como bancos de dados e dados não estruturados.

Em outra pesquisa descrita em He Hao e Zhiqing (2009), registros de incidentes são coletados em uma empresa fictícia, e é demonstrado um modelo para melhorar a eficiência no gerenciamento de defeitos, baseado no melhora da eficiência da classificação através de um classificador bayesiano. É uma abordagem para desenvolvedores e pessoas responsáveis por testes de softwares, podendo ser aplicada também no gerenciamento do serviços de Tecnologia da Informação .

Banco de Dados, Descoberta de Conhecimento Associado ao Histórico de Soluções

Na área de Bancos de Dados de Descoberta de Conhecimento, artigos utilizando a captura de conhecimento através de redes sociais, repositórios de incidentes, *blogs* e outras fontes foram encontrados, A partir dessas fontes de captura, foram aplicadas técnicas de mineração de textos e técnicas de mineração de dados em geral.

Também foram encontrados trabalhos que relacionam dados diversos utilizando técnicas de *Data Warehousing* e informação adaptativa, que muda de acordo com a variação nas ocorrências de incidentes do cliente, como descrito em (UR-RAHMAN; HARDING, 2012). Em outros artigos, como descrito em (SCHUGERL; RILLING; CHARLAND, 2008), aplicam-se as técnicas de retenção da informação e processamento de linguagem natural para fazer a mineração nos repositórios de *bugs*. Esses estudos abordam principalmente a medida da qualidade dos formulários de relatórios de *bug*s enviados pelos rastreadores de *bug*s de código aberto.

Análise de Defeitos, Determinação de Problemas e Rastreadores de Bugs (*Bug Trackers*) Associados ao Histórico de Soluções

Na área de rastreamento de *bugs*, alguns artigos demonstram que esse modelo ajuda em muito os desenvolvedores de software a conhecer qual foi o erro, resolvê-lo e aprender com isso. Porém, quando o número de *bugs* torna-se muito grande, um controle rudimentar em planilhas eletrônicas se torna insustentável, havendo a necessidade de um sistema que faça um rastreamento de *bugs* a nível do código fonte. Esse modelo tem sido usado no desenvolvimento de softwares livres como o BugZilla, como descrito em (SERRANO, 2005).

O que acontece de fato na fase de manutenção de software é que, quando um sistema é colocado em produção no ambiente do cliente, muitos outros fatores entram no processo de manutenção, como a instalação e configuração do software, a interação com outros softwares, a verificação da saúde do sistema (*health checking*), que verifica vulnerabilidades, o gerenciamento de versões (*releases*) e o gerenciamento de mudanças.

Mesmo que um software seja bem construído, se este não for instalado ou configurado corretamente, ou ainda, se houver agentes externos como memória, sistema operacional ou servidores compartilhados com outros softwares que atrapalhem o seu desempenho, poderão haver incidentes associados e, consequentemente, problemas disponibilidade.

Prevenção e Prognóstico de Defeitos de Software.

No âmbito da prevenção e do prognóstico de defeitos, vários trabalhos foram encontrados em análise estática de código, aplicando técnicas de aprendizado de máquina (*Machine Learning*), mineração de dados ou métodos estatísticos em modelos de prognóstico como, por exemplo, análise discriminante, regressão logística (Logistic *Regression*), árvores de regressão (*Regression Trees*), Vizinho mais Próximo (*Nearest Neighbor)*, Algoritmo Bayesiano, Redes Neurais e Máquinas de Suporte Vetorial (*Support Vector Machines*) (ZHANG; SHANG, 2011). Outros fazem uma análise da prevenção de defeitos ao nível do histórico de mudanças. Outros trabalhos ainda, fazem uma prevenção de defeitos ao nível dos arquivos de projeto, inicialmente a partir da localização de uma falha conhecida e fazendo um armazenamento da localização dessa falha e de qualquer mudança em conjunto ou localizações adicionadas (KIM et al., 2007).

Na prevenção de defeitos encontra-se, também, a classificação de defeitos. Foram encontrados trabalhos de classificação através do reconhecimento de padrões utilizando mineração de dados, clusterização e priorização através da severidade e da frequencia como encontrado em (PODGURSKI et al., 2003). Porém, esse modelo não utiliza de nenhum mecanismo padronizado como, por exemplo, o ODC (Orthogonal Defect Classification).

Chaves de Pesquisa Utilizadas

Com relação ao método utilizado na revisão sistemática, foram usadas palavras chaves as quais foram organizadas da mesma forma que a fundamentação teórica descrita neste apêndice, sendo que as chaves de pesquisa utilizadas estão apresentadas a seguir.

Chaves de Pesquisa para Gerenciamento de Serviços de Manutenção de Software:

- Incident Management Restoration

Portal	String de Busca
Science Direct	Incident AND Management AND Restoration AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Incident) AND Management) AND Restoration) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Incident and Management and restoration) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 9- Chaves de Pesquisa usadas nos sites de busca para Incident Management Restoration

- Time-Bounded Incident Management

Portal	String de Busca
Science Direct	Time-Bounded AND Incident AND Management AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Time-Bounded) AND Incident) AND Management) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Time-Bounded and Incident and Management) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 10 - Chaves de Pesquisa usadas nos sites de busca para Time-Bounded Incident Management

- Software Remediation SLA

Portal	String de Busca
Science Direct	Software AND Remediation AND SLA AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Software) AND Remediation) AND SLA) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Software and Remediation and SLA) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 11 - Chaves de Pesquisa usadas nos sites de busca para Software Remediation SLA

Chaves de Pesquisa para Taxionomia e Classificação de Defeitos de Software. Incluindo o ODC

- Software Defect Taxonomy

Portal	String de Busca
Science Direct	Software AND Defect AND Taxonomy AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Software) AND Defect) AND Taxonomy) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Software and Defect and Taxonomy) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 12 - Chaves de Pesquisa usadas nos sites de busca para Software Defect Taxonomy

- Software Defect Prevention Classification

Portal	String de Busca
Science Direct	Software AND Defect AND Prevention AND Classification LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: ((((Software) AND Defect) AND Classification) AND Prevention) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	((Software AND Defect AND Prevention AND Classification) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 13 - Chaves de Pesquisa usadas nos sites de busca para Software Defect Prevention Classification

- ODC

Portal	String de Busca
Science Direct	ODC AND Classification LIMIT-TO (pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (ODC) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(ODC) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 14 - Chaves de Pesquisa usadas nos sites de busca para ODC

Chaves de Pesquisa para Banco de Dados de Descoberta de Conhecimento (*Knowledge Discover Database*) para Defeitos e Incidentes:

- KDD Software Defects

Portal	String de Busca
Science Direct	KDD AND Software AND Defects AND LIMIT-TO (pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((KDD) AND Software) AND Defects) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(KDD and Software and Defects) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 15 - Chaves de Pesquisa usadas nos sites de busca para KDD Software Defects

- KDD Incident Management:

Portal	String de Busca
Science Direct	KDD AND Incident AND Management AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((KDD) AND Incident) AND Management) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(KDD and Incident and Management) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 16 - Chaves de Pesquisa usadas nos sites de busca para KDD Incident Management

Chaves de Pesquisa para Análise de Defeitos, Determinaçao de Problemas e Rastreador de *Bugs* relacionados ao Histórico de Soluções:

- Software Bug Tracker

Portal	String de Busca
Science Direct	Software AND Bug AND Tracker AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Software) AND Bug) AND Tracker) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Software and Bug and Tracker) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 17 - Chaves de Pesquisa usadas nos sites de busca para Software Bug Tracker

- Defect Causal Analysis

Portal	String de Busca
Science Direct	Defect AND Causal AND Analysis AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Defect) AND Causal) AND Analysis) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Defect and Causal and Analysis) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 18 - Chaves de Pesquisa usadas nos sites de busca para Defect Causal Analysis

Chaves de Pesquisa para Prevenção e Prognóstico de Defeitos de Software:

- Software Defect Prevention

Site	String de Busca
Science Direct	Software AND Defect AND Prevention AND LIMIT-TO(pubyr, "2014,2013,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Software) AND Defect) AND Prevention) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Software and Defect and Prevention) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 19 - Chaves de Pesquisa usadas nos sites de busca para Software Defect Prevention

- Software Defect Prediction

Site	String de Busca
Science Direct	Software AND Defect AND Prediction AND LIMIT-TO(pubyr, "2014,2013,2012,2011,2010,2009,2008,2007,2006,2005,2004,2003,2002")[All Sources(Computer Science)]
IEEE Xplore	You searched for: (((Software) AND Defect) AND Prediction) You Refined by: Content Type: Conference Publications, Journals & Magazines Topic: Computing & Processing (Hardware/Software) Publication Year: 2002 - 2014
ACM Library	(Software and Defect and Prediction) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction OR PublishedAs:magazine)

Quadro 20 - Chaves de Pesquisa usadas nos sites de busca para Software Defect Prediction

Resultados Temporais Representados em Gráficos:

A seguir são apresentados os resultados temporais obtidos na revisão, estando organizados pelas palavras chaves utilizadas,

Resultados para Gerenciamento de Serviços de Manutanção de Software:

- Incident Management Restoration

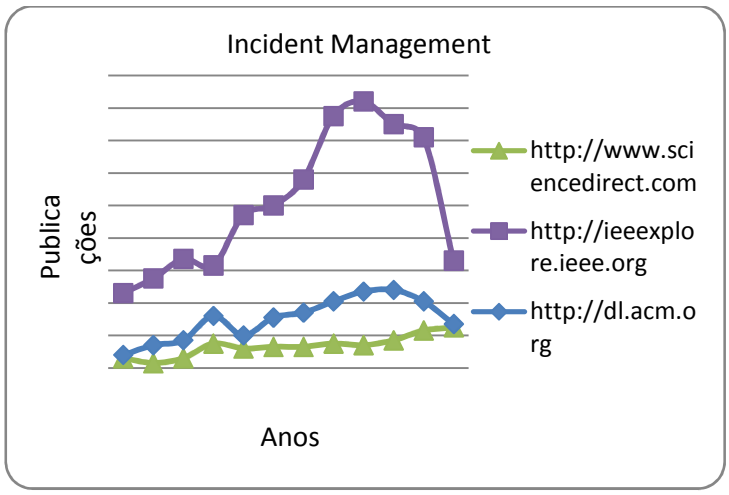

Figura 38 - Número de Publicações pela pesquisa da palavra chave Incident Management Restoration entre 2002 e 2013

- Time-Bounded Incident Management

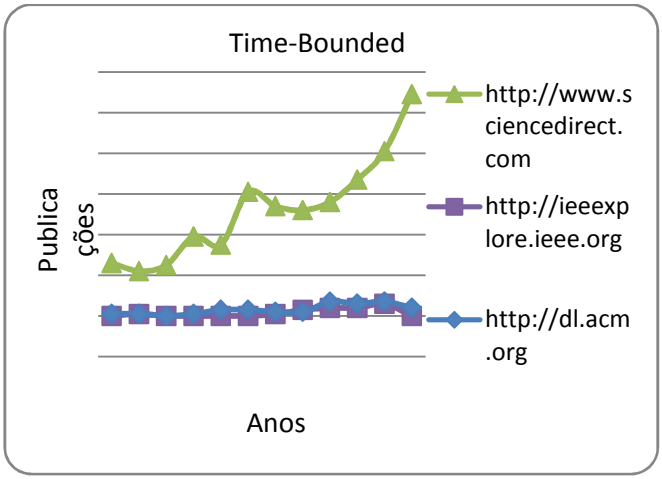

Figura 39 - Número de Publicações pela pesquisa da palavra chave Time-Bounded Incident Management entre 2002 e 2013

- Software Remediation SLA

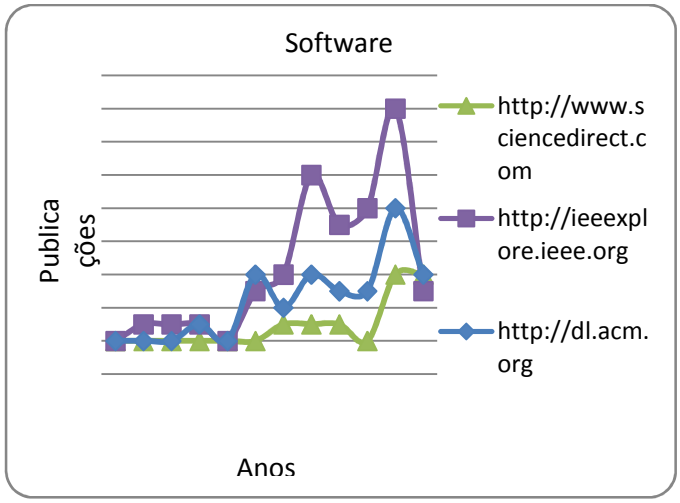

Figura 40 - Número de Publicações pela pesquisa da palavra chave Software Remediation SLA entre 2002 e 2013

Resultados para Taxionomia e Classificação de Defeitos de Software, Incluindo o ODC

- Software Defect Taxonomy

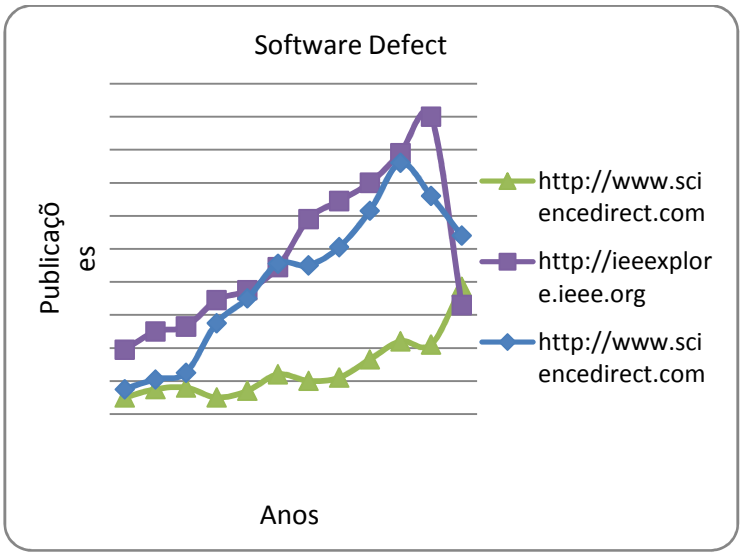

Figura 41 - Número de Publicações pela pesquisa da palavra chave Software Defect Taxonomy entre 2002 e 2013

- Software Defect Prevention Classification

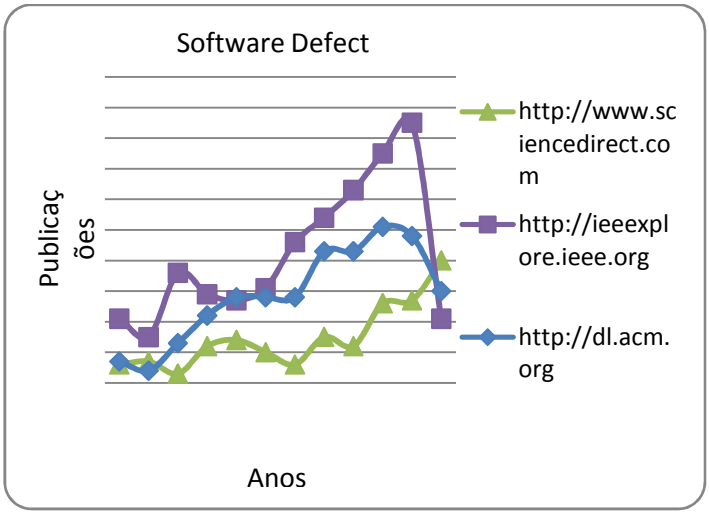

Figura 42 - Número de Publicações pela pesquisa da palavra chave Software Defect Prevention Classification entre 2002 e 2013

- ODC

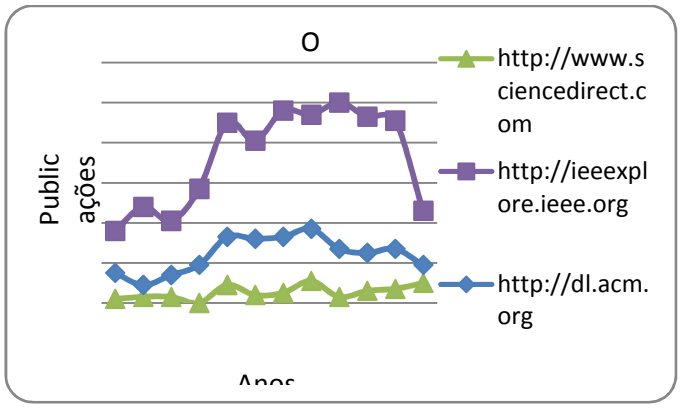

Figura 43 - Número de Publicações pela pesquisa da palavra chave ODC entre os anos de 2002 e 2013

Resultados para Banco de Dados de Descoberta de Conhecimento (*Knowledge Discover Database*) para Defeitos e Incidentes :

- KDD Software Defects

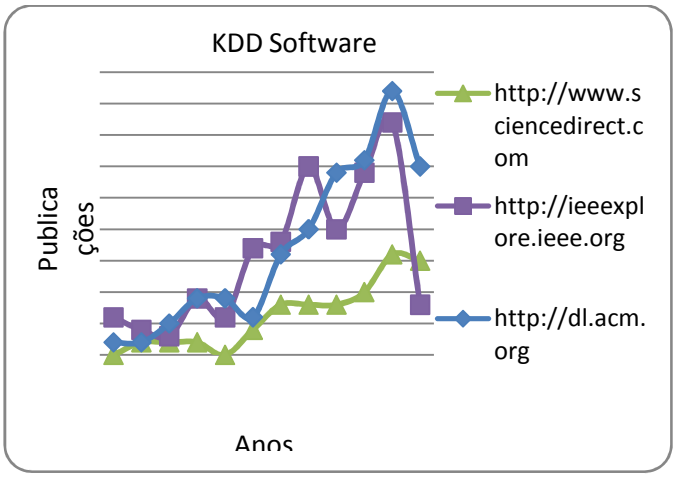

Figura 44 - Número de Publicações pela pesquisa da palavra chave KDD Software Defects entre 2002 e 2013

- KDD Incident Management

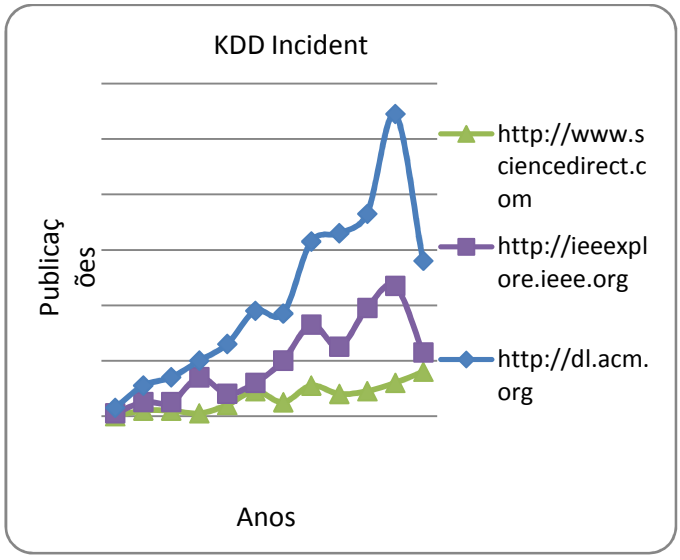

Figura 45 - Número de Publicações pela pesquisa da palavra chave KDD Incident Management entre 2002 e 2013

Resultados para Análise de Defeitos, Determinaçao de Problemas e Rastreador de *Bugs* Relacionados ao Histórico de Soluções:

- Software Bug Tracker

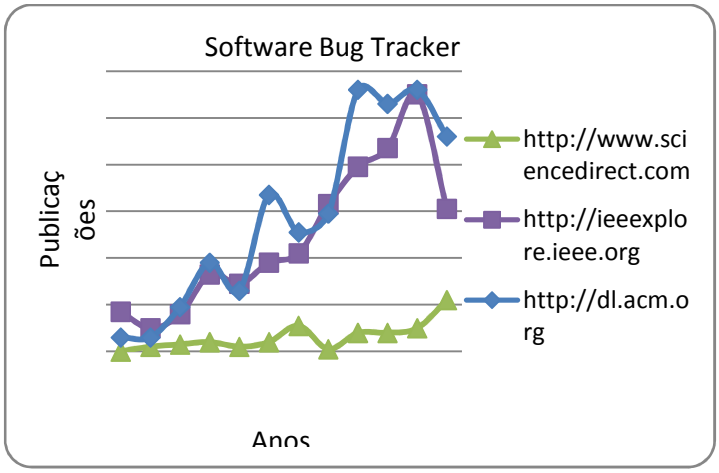

Figura 46 - Número de Publicações pela pesquisa da palavra chave Software Bug Tracker entre 2002 e 2013

- Defect Causal Analysis

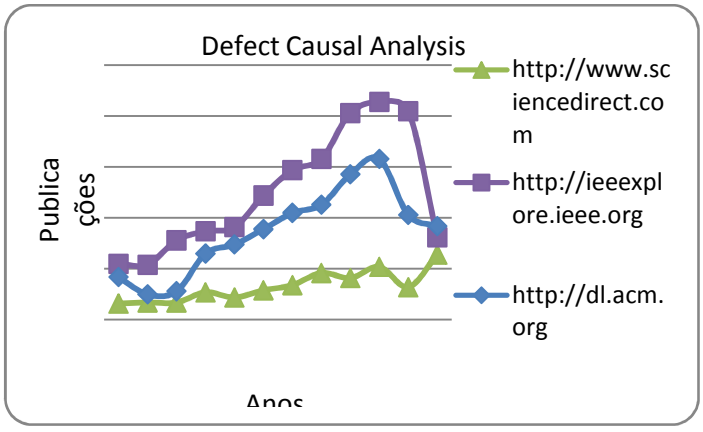

Figura 47 - Número de Publicações pela pesquisa da palavra chave Defect Causal Analysis entre 2002 e 2013

Resultados para Prevenção e Prognóstico de Defeitos de Software:

- Software Defect Prevention

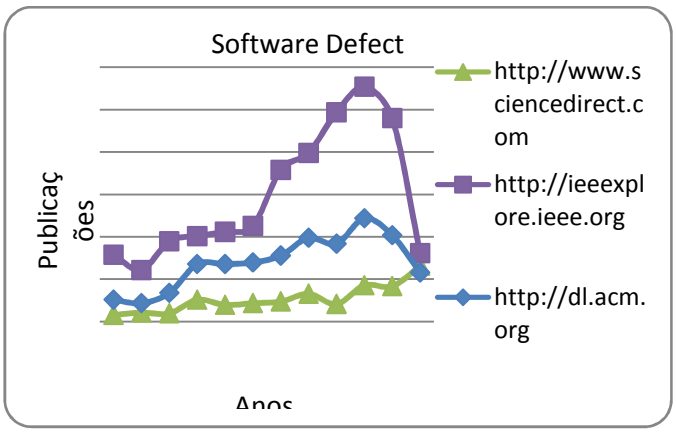

Figura 48 - Número de Publicações pela pesquisa da palavra chave Software Defect Prevention entre 2002 e 2013

- Software Defect Prediction

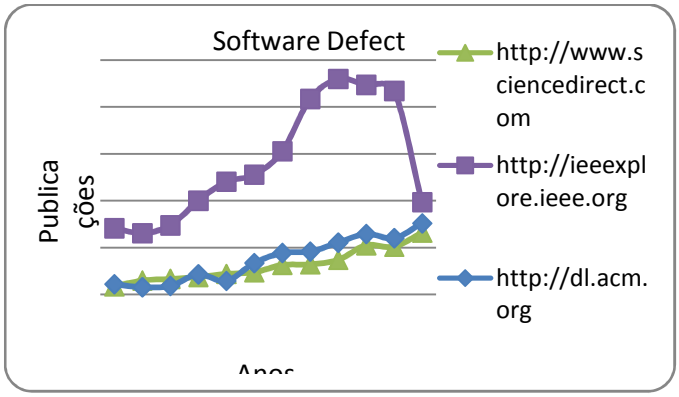

Figura 49 - Número de Publicações pela pesquisa da palavra chave Software Defect Prediction entre 2002 e 2013

Resultados da Seleção dos Artigos

Após a pesquisa das palavras chaves, selecionando 108 publicações a partir da leitura do título e do resumo, foi feita uma filtragem utilizando-se a Escala de Likert. Esta segunda avaliação foi realizada lendo-se a introdução, a conclusão e em alguns casos toda a publicação Segue o resultado da análise a partir da Escala de Likert (-2 a +2) para as 108 publicações selecionadas iniciamente, em termos de percentagem e número de artigos de acordo com a Figura 50.

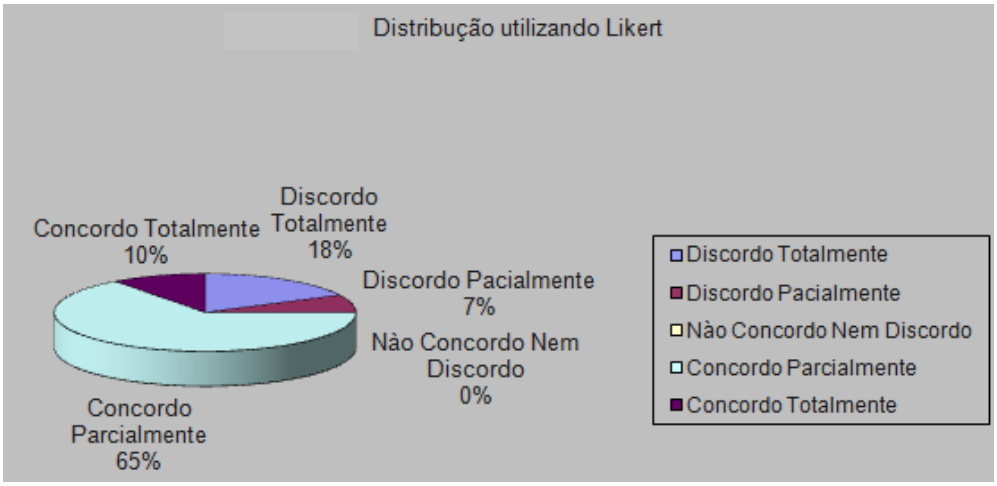

Figura 50 - Percentagem das Publicações de acordo com a divisão de relevância pela escada de Likert

A partir das 108 publicações selecionadas na pesquisa por palavras chaves, foram selecionadas as 81 publicações que tiveram nota de 1 a 2 na escala de Likert, conforme resumido no Quadro 20.

Descrição	Valores
Discordo Totalmente (-2)	19
Discordo Pacialmente (-1)	8
Não Concordo Nem Discordo (0)	0
Concordo Parcialmente (1)	70
Concordo Totalmente (2)	11
Total Selecionado ((1) + (2))	**81**

Quadro 21 - Número de publicações classificadas pela escala de Likert

Portanto 81 publicações foram utilizadas para a revisão sistemática. A Figura 46 descreve o processo completo de seleção das publicações.

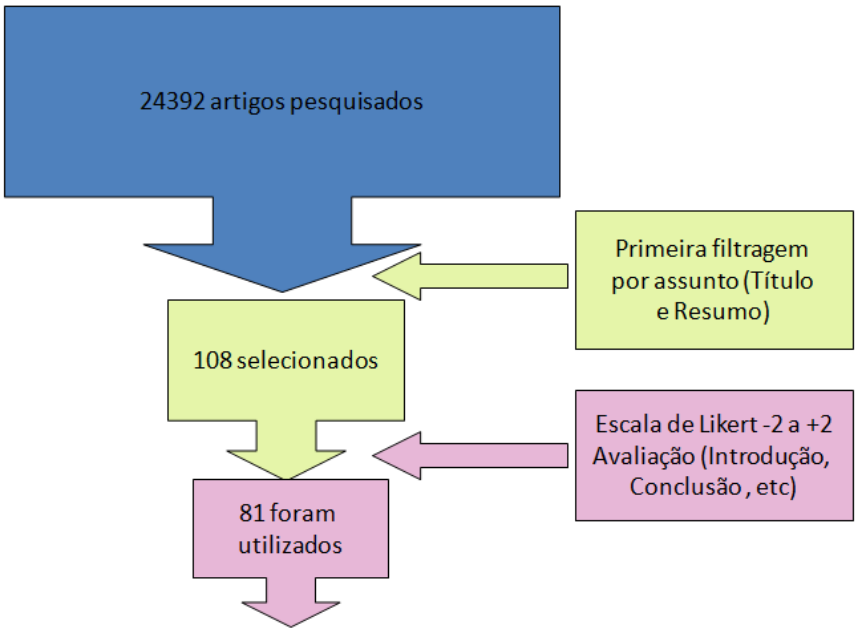

Figura 51 - Ilustração da filtragem de artigos desde a pesquisa inicial até o selecionamento final de artigos relevantes.

Resultados da publicação por países

As publicações distribuídas por países apresentam os países dos autores principais dos artigos. Os resultados estão apresentados na Figura 47.

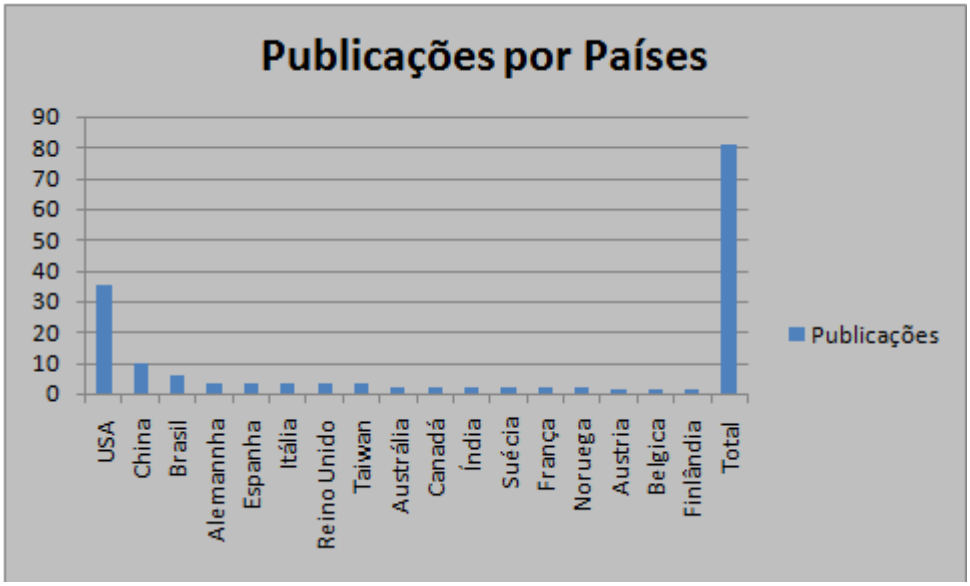

Figura 52 - Divisão de publicações por países em relação aos artigos selecionados

APÊNDICE B – ARTIGO PUBLICADO

Classifying defects in software maintenance to support decisions using hierarchical ODC

.Marcelo M. Manhães, Maria Claudia P. Emer, Laudelino C. Bastos

Departamento de Informática – Universidade Técnica Federal do Paraná (UTFPR)

Caixa Postal 80230-901 – Curitiba– PR – Brazil

marcelo.manhaes@hotmail.com, mclaudia@dainf.ct.utfpr.edu.br,
bastos@dainf.ct.utfpr.edu.br

Abstract. *Software maintenance is a critical part in software life cycle where fast production deployment and effectiveness for high quality software are pressure factors. For cloud and regular Service Providers such as out sourcing enterprises it is a key factor of success due to service level agreements closed to customer needs. Although, more high level tools of system monitoring are available into market to increase high availability, a lot of software packages go to service providers with fatal bugs. The effectiveness in terms of service components failure identification and priorities definition focused on incidents information are few used. This paper presents a method to extend orthogonal defect classification (ODC) in terms of triggers and sources attributes in a hierarchical way to improve technical support decisions in service providers. This approach classifies better errors on this phase using service components in software maintenance from incidents history. The ODC original triggers and sources are decomposed in values closely related to software maintenance service components without change the orthogonality essence of original ODC using a drill-down feature. These attributes are used with ODC defect type and ODC impact in a multi-dimensional modeling. This method reaches better service errors identification, remove it earlier than standard methods and better process feedback to manage service provider decisions. The method suggests an improvement in service providers functionality based on experiments.*

1. Introduction

In service providers today, there are several cases where customer software is inserted in a selected infrastructure service, for instance, in a cloud environment. Discover where problem is found becomes very important to drive next release tests and directing problem to customer's software or infrastructure or still if the problem is clearly detected to customer code, service provider can offer a refactoring code service. It is a fundamental point where software changes have huge frequency to follow time to market.

Quality and cost are key points to guarantee success of service providers where rely on factors such as component and board quality, efficient application deployments, high availability, a good support of system diagnostics and behind these factors a process that controls quality assured, resource and costs [Toai et al. 2006]. To achieve these key

performance indicators a correct and functional classification of errors improve a correct prioritization and hence a faster resolution. The information about errors would be valuable to classify and prioritize service components in software maintenance or when software goes to production.

In classification of defects field several studies were described using flat classification as described in [Knuth 1989] or hierarchical model in [Beizer and Vinter 2001]. In [Gray 1986] concept of activations in software production failures were introduced based on software tested and used for a long time defining type of bugs in two categories transients (easily reproduced) and no-transients (not easily reproduced).

[Chillarege et Al., 1992] introduced ODC (Orthogonal Defect Classification) that turned a rich and multi-faceted technology that provides many benefits. One of them is the prevention of defects as decreasing the number of defects being injected into design and code [IBM 2013]. There were some works related to improve some test phases where ODC was extended to accomplish this improvement as described in [Li et al. 2010] for black box testing and for code inspection as described in [Kelly 2000].

Defect classification varies according to different purposes, but on this paper the goal is presents an ODC framework to software maintenance, purposing an extension of ODC classification for it called ODC-SC (ODC Service Components). This method is based on fact that in production there is a huge volume transaction. Moreover software maintenance is an important slice in software development process with high costs related. All classification is extracted from incident tickets and applied techniques of multi-dimensional modeling and drill-down in some ODC attributes.

Classification and prioritization are key points to follow continuous and high volume change process where service providers need to provide an efficient defect prevention process in order to follow a huge demand for business agility. Also it is fundamental to provide good measures about production environment and if a problem is from customer code or if is from infrastructure provisioning and how to improve for deployments.

The main purpose of ODC-SC method is described as follows: 1) provides service providers managements and outsourcing enterprises controls with a comprehensive analysis method to define if errors are infra-structure based or from customer data or code based. 2) Provides infra-structure prioritization of defects in order to inspect defects types for

individual components or still to prioritize class of problems during reviews. 3) Help new deployments to improve high availability in production.

This paper is organized as follows. Section 2 presents a discussion on related works. Section 3 proposes a method to increments ODC triggers and ODC source values for software maintenance using a service components concept. Section 4 provides an experiment to validate of method ODC-SC from software maintenance records. Discussions about some decisions are explained in section 5. Some conclusive remarks are given in section 6.

2. Background and related work

It was done a bibliographic review around 11 years (2002 to 2013) for journals on reference sites Science Direct (http://www.sciencedirect.com/), ACM Library (http://dl.acm.org/) and IEEE Xplore (http://ieeexplore.ieee.org) for similar or related works. From initial articles, past studies were gathered before the initial time frame. In classification of defects field, several studies were described. These studies in literature start by one of the initial materials on this field in [Knuth 1989] where it was designed from experiences in projects of medium size about an evolution of a typesetting system using a flat categorization mixing faults with enhancements. Studies described in [Beizer and Vinter 2001] provided a comprehensive schema for software faults categorization using a hierarchical model with 10 initial categories and 100 leaf categories in software development process. These initial works contributed on software fault categorization, however, provided a model with high difficulties to classify due to ambiguities in several software faults becoming difficult to use.

Other studies produced another approach focused in report of fails (activations) that affected availability and reliability when software was in production [Gray 1986]. The state production was considered when software has been tested and used for long periods of time. On referred research it was introduced concepts of **Heinsenbugs** or no deterministic faults where it was related to fails not reproduced with repeated execution and **Bohrbugs** or deterministic faults where bugs are reproduced easily. This study concluded most of production software faults were transients. However software introduced in production was not considered on related study.

The term ODC was first described in [Chillarege et Al., 1992] with a proposal to classify defects to improve software development life cycle process. The main idea would be extract semantic information from defects. From defect information it was extracted a relationship between cause and effect. Still in [Chillarege et Al., 1992] the natural extension

of ODC is the defect prevention process where fits very well with defect data collected from stability or system tests where provides most accurate information requirement for the software quality assessment and if is classified well can focusing in priorities and also in defect fix effectiveness.

Related works as described in [Li et al. 2010] shows a method called ODC-BD that is a black box testing using ODC. On this solution, ODC is totally customized to black-box expanding the defect attributes and describing specific defects and therefore improving effectiveness as research says. In other article, in code level scope, it was proposed a new defect classification from original ODC that was called ODC-CC (Orthogonal Defect Classification Computational Codes) to be used for code level defects [Kelly 2000].

Similar approaches from this research are described in [Podgurski et al. 2003] where fails that are caused from same defect and have a considerable number of occurrences can be elected to defect prevention according to fail criticality. The method used is a manual clustering to define similar fails in a bi-dimensional space where the points of failures are positioned. The similarities of fails are measured from the distance between then according specific criteria. Group of failures can be identified from a graphic visual inspection. The users can judge what fails are more related instead of believe in automatic clustering method. A disadvantage of this model is that only 2 dimensions are exposed and little differences between failures are poor presented. ODC can improve this approach because it is a multi-dimension classification of defects.

Recent related studies in other areas using ODC was found such as garment industry it can be highlighted a quality management system using an Online Analytical Processing and mining association rules. This system extracts garment defects through hidden patterns in order to work on defect prediction, root cause identification and proactive actions to improve quality. The quality improvements are reached by fix what steps on production process raise more failures [Lee et al. 2013].

Based on this state of the art cited above, with few studies focused for software maintenance with respect to classification and analysis of defects, but a lot of studies related to ODC, the ultimate goal of this paper is to optimize the ODC defect classification for software maintenance to allow support decisions to define what the scope of problems and defect prioritization to improve maintenance process.

3. Classification schema for software maintenance

The method used on this research keeps the original structure of ODC and componentize ODC triggers and ODC sources. This will change initial flat characteristic of ODC putting a hierarchical mode. Also it can be considered a drill down method of ODC triggers and ODC sources. The information about what attributes make in hierarchical mode were motivated from incident information gathered from several customers supported in Enterprise "A".

The Enterprise "A" is a big player in software outsourcing where manages hardware and software located in other enterprises such as banks, cred card holdings, electronic enterprises, retail business and so for. If is counted only middleware products supported from a local team, it reaches around 20 customers. The support data is shown in a central tool to control all work tasks. The available defect´s data history shows a high difficulty to avoid ambiguity in report issues and also a difficulty in prevent issues where several products are involved such as databases, middleware, operational system and hardware types. Also the volume of incidents are high, around 2000 by month if counts all middleware products from customers where Enterprise "A" manages. The hardware and software sets have high complexity configurations and they are mixed in multiple combinations and versions as customer environment is completely third part managed.

Before the deeper classification definition, it will be defined a big ODC picture regarding a defect report. To define a defect report from an incident, there are four main attributes that it will focus in ODC as described below (Figure 1) [Chillarege 2006].

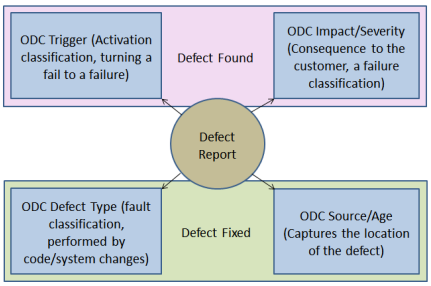

Figure 1. A defect report formed by ODC trigger and ODC impact when open a defect and ODC defect type and ODC Source/Age when close a defect [Chillarege 2006].

ODC triggers are very important because can show a classification of defect activation [Chillarege 2011]. ODC defect type measures the software development process where defects are fixed regarding faults representing what correction was applied

[IBM 2013]. ODC impact represents the failure classification to the customer view. ODC source indicates the location of defect [Chillarege 2006].

Based on these four main attributes in a multi-dimensional mode and based on data gathered from incidents in Enterprise “A”. It can be defined a general mapping as described on Figure 2.

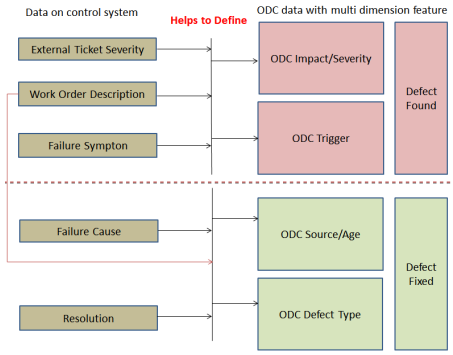

Figure 2. Mapping from original support data and ODC attributes in a multi-dimensional feature.

The initial mapping consists to map External Ticket Severity to ODC impact and Resolution to ODC Defect Type. Work Order Description and Failure Symptom map to ODC trigger and Impact. Finally, Work Order Description, Failure Cause and Resolution map to ODC Source. These mappings can be generalized to other service providers, considering that most of incident´s report has at least severity, description of incident (work order) and resolution. An important point to mention is that from the four ODC attributes is that ODC trigger and ODC source will have a drill-down feature. On the other hand, ODC impact is mapped from numbers (1 to 4) and ODC defect type are mapped directly from operational variables source data based on some criteria.

When we classify field defects, on drill-down feature, trigger is the first point, by selecting the trigger that best represents the environment or condition that exposed the defect in the customer’s or service provider´s environment. The activity is selected based on the task associated originally to the trigger [IBM 2013].

Software maintenance was a base for this research but based on closely activity that is system test. These are the triggers in ODC scope [Chillarege and Ram 2002]:

1. Workload Volume/Stress: it is related to the limits of performance, resources, users, queues, traffic and so for.
2. Recovery/Exception: Invoke exception handling, recovery, termination, error percolation for instance.

3. Startup/Restart: Relevant events of turning on, off or changing the degree of service availability.
4. Hardware Configuration: Incidents surfaced as a consequence of changes in hardware setup.
5. Software Configuration: Incidents surfaced as a consequence of changes to software setup.
6. Blocked Test/Normal Mode: Customer found nonspecific trigger where test could not run during production deployment.

On software maintenance phase, the environment can be so complex and trigger mitigation is a fundamental point to find a problem as soon as possible and provide a deeper classification and mitigation of risk and focus on priorities. This demand is being a fundamental point to enterprise that provides software services such as web site hosts, public or hybrid cloud services where applications that are deployed on the service providers need to deliver a fast resolution for problems. Based on this demand triggers subtypes are spread around ODC triggers as described in Table 1.

Table 1. The table is a proposal of trigger sub types for software maintenance and verification with original ODC triggers associated.

Trigger sub type	Related to what Original Trigger?	Description
Application	Software Configuration, Workload Stress, Recovery Exception, Startup/Restart	Application Code that belongs to customer or any third part code. It is related to any application code error, configuration of application on the system.
Security Infrastructure	Software Configuration, Workload /Stress, Recovery Exception, Startup/Restart	Errors related to certificates (hardware or software), firewalls, data access, Single Sign On (SSO), authorization and authentication.
CPU	Hardware Configuration, Workload /Stress	Triggers related to CPU problems caused by low capacity sizing or physical problems.
Memory	Hardware Configuration, Workload /Stress	Triggers related to Memory problems caused by low capacity sizing or physical problems.
Storage	Hardware Configuration, Workload /Stress	Problems closely to bad dimension of file system or some hardware problem. Application code can generate a lot of logs files raising File system full.
Other Hardware	Workload /Stress	Hardware that is not storage, network, print services or security infra.
Network	Hardware Configuration,	Problems closely related to hardware firmware

	Workload /Stress, Recovery Exception, Startup/Restart	problem, network sizing or other that are not CPU, Memory, Storage, printing services for instance.
Printing	Software Configuration, Hardware Configuration, Workload /Stress, Recovery Exception, Startup/Restart	Printing problems related to hardware or software drivers involved.
OS	Software Configuration, Workload /Stress, Recovery Exception, Startup/Restart	Problems related to incorrect configuration and patching (version update), slow response related to capacity, resource contention, file system settings, ulimits settings for UNIX . In short all OS settings.
Middleware	Software Configuration, Workload /Stress, Recovery Exception, Startup/Restart	All configuration problems related to these products such as application servers, main frames, message queue servers products, HTTP servers, Web servers and so for.
Database	Software Configuration, Workload /Stress, Recovery Exception, Startup/Restart	All problems related to database products itself or any data access configuration such as drivers.
Unknown Errors	Blocked Test/Normal mode	There is not enough information to define where the issue occurred such as when a migration is in place and new software version is not working.

It is very useful for software maintenance to determinate what component in service provider is causing more problems regarding availability in production. This conclusion is based on incidents report from Enterprise “A”.

From Table1 described earlier it is provided a distribution of software and hardware components gathered from service providers in Figure 3 where is it shown the distribution along the ODC base system test triggers. Again this information is collected from incident reports on Enterprise “A” where shows the components involved.

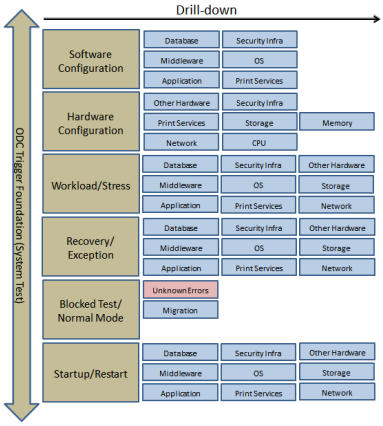

Figure 3. Components in service providers distributed in foundation ODC triggers using a drill-down technique

Another point where ODC attributes will be deeper on this work is related to ODC source that describes where defect has been found when fixed. It can be considered a mitigation of ODC triggers where describes the surface of fail.

It will be used the ODC foundation source values in the first level. In the second level the parameters will be imported from ODC triggers subtypes to form second level of ODC sources as described in Figure 4. This approach guarantees the tight relationship with surface when defect is found with high level location when defect is fixed.

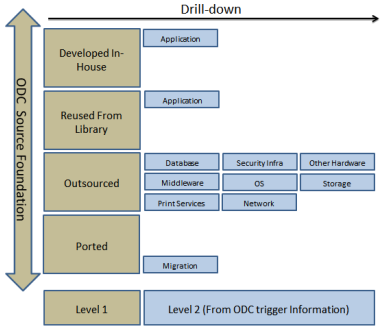

Figure 4. ODC triggers being placed as second level for ODC source attribute

In the third level these values are expanded in sub components related to products or configuration components and following same source data in Enterprise "A". This third level can be related to area where it needs to focus such as support for test service providers or software development. On this level describes in depth where it can be found the problem when was fixed. The information about what and how was fixed is described in ODC defect type. The Figure 5 explains the third level for ODC source. It can be use optionally depending on complexity of maintenance services. Despite third level into ODC source necessary or not, the important point is to position service components into ODC

source foundation for instance, developed In-House and Reused From library (Customer) or Outsourced (Service Provider). This point can be very important to define responsibilities in a regular or cloud service provider.

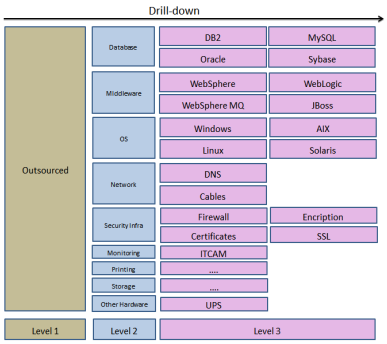

Figure 5. Third level of ODC sources related to specific products or components

4. Validation of ODC-SC

The method was applied from service work orders extracted from different teams such as database, middleware, network main frame teams and others in the outsourcing Enterprise "A". This data set was delivered in comma-separated values (CSV) format. Indeed, it was applied a manual classification at first phase to understand better business case. Using a set of 50 entries of work orders that could be considered a training set for future data mining purposes. From method it was defined four ODC attributes: trigger, impact, and source and defect type. Besides original ODC, it was applied service components (drill-down) into triggers and sources. With final data, using a simple graphic tool, such a Microsoft Excel, it shows the result as described on Figure 6.

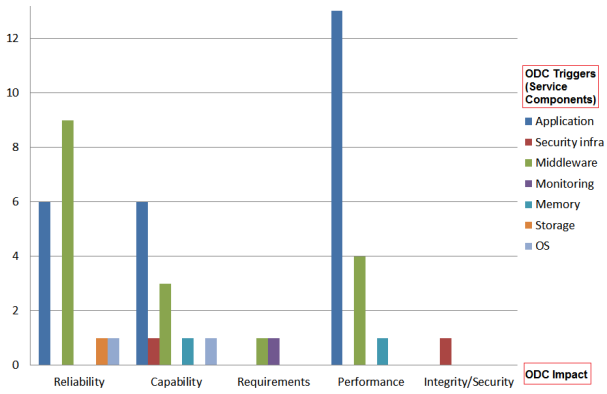

Figure 6. The data classified in a graphic

This two dimension graphic is extracted from tickets in Enterprise "A" and shows the distribution of errors using ODC triggers with service components feature, axis y. It

was used a kind of analysis dice between ODC impact attribute and ODC trigger, with trigger service components drilled down on this experiment. These service components on axis y are distributed around axis x (Impact ODC attribute). The impact ODC represents what was impacted on customer environment. The conclusion on this chart suggests that application code is impacting of system in a considerable value (Performance, Reliability and Capability). The customer code needs to be improved in performance. Reliability in original ODC represents when system is down by a general failure and performs high availability impact. Capability in ODC means that the system is being impacted partially in some functions. On the other hand, Middleware from service provider is impacting system reliability in high levels and needs to be improved.

5. Discussion

As defect is closed in software maintenance, service components are included on second level of ODC source making easier to discover defects in complex environment of service providers, increasing defect fix hate by time. With this information, a prioritization of defects can be executed from customer impact, for instance, defining fails surfaced by application code and middleware as maximum priority. The solution can be expanded for other service providers in software maintenance since incident data being accurate reported with description, impact and resolution data as minimum required. The third level on ODC source is optional but can improve defect prevention.

6. Conclusion and Future Work

This research works better ODC triggers and sources drill-down classification measures to improve effectiveness in software maintenance. Also works with ODC defect type and ODC impact directly in a multi-dimensional way. With this method it is easier to define priority of fixes and can be a guarantee that software will have a better quality assured using process feedback feature of ODC. Also with an ordinated and multi-dimensional data it is possible to fill weakness in service providers such as better resources provisioning and defect prevention for future deployments such as application code. The solution is expansible to regular or cloud service providers. In cloud providers for instance it have a high number of used could services (private or public) to be used by customer. Decide if customer or service provider is problem cause is a key point to fix defect rates. About future work, the method can be improved to drill-down ODC impact in order to define what business applications are more important to fix for incidents and provides defect prevention prioritization work.

7. References

Beizer, Boris, Vinter, Otto (2001) “Bug taxonomy and statistics. Technical report”, Software Engineering Mentor, p. 2630.

Chillarege, R., (2011) "Understanding Bohr-Mandel Bugs through ODC Triggers and a Case Study with Empirical Estimations of Their Field Proportion," Software Aging and Rejuvenation, 2011 IEEE Third International Workshop on , vol., no., p.7-13

Chillarege, R., (2006) “ODC - a 10x for Root Cause Analysis”, Proceedings. RAM 2006 Workshop.

Chillarege, R., Ram Prasad, K., (2002) "Test and development process retrospective - a case study using ODC triggers," Dependable Systems and Networks. Proceedings. International Conference on, vol., no., p. 669-678.

Chillarege, R., Bhandari, I. S., Chaar J. K., Halliday, M. J., Moebus, D. S., Ray, B. K., Wong, M. (1992) “Orthogonal defect classification-a concept for in-process measurements”, IEEE Transactions on Software Engineer, EUA, vol. 18, p. 943-956.

Gray, J. (1986) “Why do computers stop and what can be done about it?”, In Proceedings of Symposium on Reliability in Distributed Software and Database Systems (SRDS-5). IEEE CS Press, vol. no., p. 3–12.

Kelly D., (2000) “An experiment to investigate a new software inspection technique”, Master’s Thesis, Royal Military College of Canada.

Knuth D. E. ,(1989) “The errors of TEX. Software–Practice and Experience”, V. 19, Issue 7, July 1989, p. 607-685.

Lee, C.K.H., Choy, K.L., Ho, G.T.S., Chin, K.S., Law, K.M.Y., Tse, Y.K. (2013) “A hybrid OLAP-association rule mining based quality management system for extracting defect patterns in the garment industry”, Expert Systems with Applications, vol. 40, Issue 7, 1 June, p. 2435-2446.

Li, N., Li, Z. and Sun, X. (2010) “Classification of Software Defect Detected by Black-box Testing: An Empirical Study”, Second WRI World Congress on Software Engineering. China, vol. 2, p. 234-240.

IBM research center for Software Engineering (2013) “Examples of the Use of ODC”, http://www.research.ibm.com/softeng/ODC/ODC.HTM, March.

Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M. (2003) "Automated Support for Classifying Software Failure Reports." In: Proceedings of the 25th International Conference on, vol., no., p. 465-475.

Toai, V., Zhiyuan W., Eaton, T., Ghosh, P., Huai L., Young L., Weili W., Rong F., Singletary, D., Xinli G., (2006) "Design for Board and System Level Structural Test and Diagnosis," Test Conference, ITC '06. IEEE International, vol., no., p.1,10, 22-27.