Análise da Relação entre Chamados e Modificações no Código-Fonte de um Sistema de Acompanhamento de Obras

Luciano Antônio Cordeiro de Sousa, Bruno Santana da Silva

Instituto Metrópole Digital
Universidade Federal do Rio Grande do Norte
Av. Senador Salgado Filho, 3000 – 59078-970 – Natal – RN – Brasil

lacsousa@gmail.com, bruno@imd.ufrn.br

Abstract. *Some systems have a long maintenance period, being modified for functionality adequacy and bugs fix. After some time, it is dificult to answer: Did bugs arise during system development or maintenance? What modification in the system could have caused a certain bug? Questions like these had already some answers in academy, but not always professionals understand how these questions may affect their work in industry. This paper presents a case study about relations between bug reports and modifications in source-code over 2 years of a construction monitoring system of an energy company. Understanding these relations contributes to professionals' awareness about opportunities for improvement in their practice during development and maintenance process of systems.*

Resumo. *Alguns sistemas passam por um longo período de manutenção, sendo modificados para adequação de funcionalidades e correção de falhas. Depois de algum tempo, pode ser difícil responder: As falhas surgiram durante o desenvolvimento ou durante a manutenção do sistema? Que modificação no sistema poderia ter causado determinada falha? Perguntas assim já encontram respostas na academia, mas nem sempre profissionais compreendem com elas podem afetar seu trabalho na indústria. Este artigo apresenta um estudo de caso sobre relações entre chamados e modificações no código-fonte ao longo de 2 anos de um sistema de controle de obras de uma empresa de energia. A compreensão dessas relações contribui para conscientização de profissionais sobre oportunidades de melhoria na sua prática durante o processo de desenvolvimento e manutenção de sistemas.*

1. Introdução

Muitos fatores influenciam a qualidade de um sistema, como processo, tecnologias, condições de desenvolvimento (custos, prazos, etc.) e as pessoas envolvidas [Bourque e Fairley, 2014]. Depois que um sistema é implantado, ele entra na fase de manutenção [Grubb e Takang, 2003] onde ocorrem correções de falhas, bem como inclusões, remoções ou modificações de funcionalidades. A manutenção de alguns sistemas pode durar por anos. Nesse período, as modificações no código podem afetar a qualidade do sistema, positiva ou negativamente.

Apesar de a academia considerar fundamental manter uma preocupação com a qualidade durante todo desenvolvimento e manutenção do sistema [Bourque e Fairley, 2014], alguns profissionais na indústria ainda não consideram isso fundamental e negligenciam práticas que garantam a qualidade, como testes [Craig e Jaskiel, 2002; Delamaro et al., 2007], por exemplo. Não é raro encontrar profissionais que tratam testes de software com pouco cuidado, argumentando ser assunto apenas acadêmico ou não condizente com a agilidade do projeto. Esses profissionais ainda desconhecem os benefícios de realizar testes adequadamente, e os malefícios de ignorá-los, durante sua prática profissional. A compreensão do que acontece em casos reais poderia contribuir para uma **conscientização desses profissionais**.

Este trabalho apresenta os resultados de uma pesquisa exploratória [White e Roth, 2009] que realizou um estudo de caso [Stake, 1995] sobre o Sistema de Acompanhamento de Obras (SAO) para identificar relações entre chamados de correção e modificações no código-fonte, no período de dezembro de 2012 a março de 2015. Essas relações (ou ausência delas) contribuem para compreender algumas consequências de decisões tomadas e identificar oportunidades de melhorias no processo de desenvolvimento e manutenção do sistema de informação analisado.

Uma empresa de energia presente em vários estados no Brasil desenvolveu o SAO para auxiliar a gestão de informações sobre suas obras de engenharia. Quando algum problema for encontrado, ela deve tomar ações corretivas para dar andamento às obras; evitando atrasos, retrabalho e desperdícios desnecessários. O SAO foi implantado com grande expectativa. Logo no início, os usuários foram expostos a muitas falhas no sistema, algumas em funcionalidades fundamentais para o negócio. As dificuldades foram tão significativas que eles cogitaram abandonar o uso do sistema. A reputação da equipe de TI também foi afetada negativamente, pois deixaram a impressão de que o investimento na construção do SAO não resultou em um produto relevante.

2. Estudo de Caso

O estudo de caso foi realizado em quatro etapas sequenciais que investigaram o SAO no período de dezembro de 2012 a março de 2015 [Sousa, 2016]. Na primeira etapa, investigamos funcionalidades e aspectos técnicos do SAO para definir um contexto de análise. Na segunda, analisamos o histórico de ocorrência dos chamados de correção e o histórico de criação de testes automatizados (código-fonte de teste). Na terceira, investigamos a influência da ocorrência dos chamados sobre a criação dos testes automatizados, e vice-versa. Até este ponto, foi possível identificar oportunidades de melhoria no processo de desenvolvimento do SAO, mas não explicar por que tantas falhas chegaram os usuários. Este artigo apresenta a quarta etapa deste estudo de caso, que aprofundou a investigação analisando as modificações no código-fonte do sistema.

3. Objetivo

Poucas macrofuncionalidades foram referenciadas de forma recorrente em vários chamados. As macrofuncionalidades "Manter RDF", "Manter QCO" e "Gerar RDO" concentraram o maior número de chamados (61% do total) no período. O objetivo da quarta etapa do estudo foi identificar oportunidades de melhoria no processo de desenvolvimento e manutenção do SÃO através da análise das relações entre chamados

de correção e modificações no código-fonte para estas três funcionalidades: **Quais problemas relatados nos chamados podem ter surgido durante o desenvolvimento do SAO? Quais podem ter surgido durante sua manutenção?** Se um chamado puder ser relacionado a modificações no código-fonte realizadas durante a manutenção, a falha relatada pode ter origem na correção de outras falhas ou na evolução das macrofuncionalidades. Por outro lado, se não houverem modificações no código-fonte relacionadas com um chamado durante a manutenção, provavelmente a falha nele relatada teve origem ainda durante o desenvolvimento.

4. Metodologia

A identificação das macrofuncionalidades foi realizada com a leitura do documento de visão e de especificação de requisitos do SAO. Elas puderam ser detalhadas em **funcionalidades** mais específicas para facilitar a identificação das relações entre chamados e modificações no código-fonte (Tabela 1).

Tabela 1. Detalhamento das macrofuncionalidades com mais chamados.

Manter RDF	Manter QCO	Gerar Relatório Detalhado do RDO
Incluir RDF, Copiar RDF, Consultar RDF, Alterar RDF, Excluir RDF, Concluir RDF, Reabrir RDF e Buscar RDF	Incluir QCO, Consultar QCO, Comentar QCO, Validar QCO, Alterar QCO, Excluir QCO, Responder QCO, Avaliar Resposta do QCO e Buscar QCO	Gerar relatório

Os **chamados** coletados na segunda etapa do estudo de caso foram o ponto de partida nesta quarta etapa. Dos chamados que apontam para correções no SAO, apenas 59 estão relacionados com as três macrofuncionalidades de interesse. Dois destes chamados foram desconsiderados por se tratarem de problemas de conexão e problemas de *layout*, difíceis de relacionar com modificações no código-fonte. Deste modo, nesta etapa consideramos 57 chamados, cada um com identificação, data, descrição do problema feita pelo usuário e observações do desenvolvedor sobre sua resolução. Com base nesses dados originais dos chamados, estruturamos manualmente outros dados que permitem relacionar o chamado com mudanças no código-fonte através das funcionalidades. Para cada chamado definimos:

- parte_sistema: indica a parte do sistema que sofreu alteração com o atendimento do chamado, por exemplo: código-fonte, banco de dados ou perfil de acesso;
- ações_funcionalidades: indica as ações das funcionalidades referenciadas no chamado, por exemplo: buscar, consultar, incluir, etc.;
- objeto_funcionalidades: representa o objeto manipulado pelas funcionalidades, por exemplo: RDF, QCO e RDO; e
- partes_afetadas_objeto: como muitos dos objetos do domínio são composição de outros, este campo descreve as partes do objeto referenciadas no chamado, por exemplo: paralisação, mão de obra e permissão de trabalho são partes do RDF.

As **modificações no código-fonte** do SAO têm sido controladas por um servidor Apache Subversion. Para cada mês analisado, todo código-fonte da aplicação foi copiado (*checkout*) e armazenado num diretório diferente. Depois, as mudanças mensais

no código foram identificadas através de uma comparação automática de arquivos feita pelo WinMerge. Todo o código fonte de cada mês foi comparado com o do mês anterior. Sempre que o WinMerge encontrou alguma diferença entre arquivos, a comparação das versões do código-fonte foi salva em um arquivo PDF que destacou em amarelo as linhas modificadas, conforme ilustrado na Figura 1. À esquerda encontra-se o código de um mês, à direita o código do próximo mês.

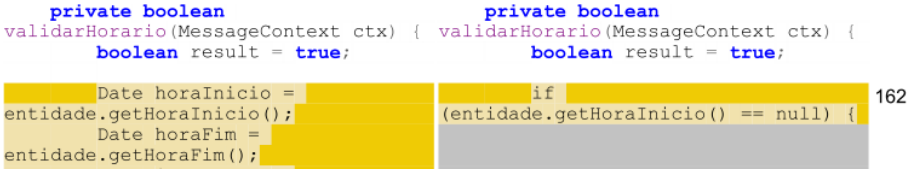

Figura 1. Comparação de código feita pelo WinMerge.

Foram identificadas 426 modificações no total. Cada modificação foi registrada manualmente com os seguintes dados, além de um identificador:

- tipo_modificação: indica o tipo da modificação que ocorreu no código-fonte; por exemplo: incluir, editar e excluir linhas de código;
- parte_código: indica a parte do código-fonte que sofreu a modificação; por exemplo: variável, método, classe ou interface com usuário;
- ações_funcionalidades_afetadas: representa as ações das funcionalidades afetadas pela modificação do código-fonte, por exemplo: buscar, consultar, etc.;
- parte_afetada_objeto: representa a parte do objeto afetada pela modificação no código-fonte. Por exemplo, se uma modificação afetou mão de obra de um RDF, este campo indicaria apenas mão de obra;
- pacote: indica o pacote que o código-fonte faz parte, por exemplo: domínio, apresentação, serviço, visão; e
- data: a data do último dia do segundo mês da comparação, no formato YYYY-MM-DD. Por exemplo, na comparação de maio com junho de 2014, a data foi 2014-06-30.

As funcionalidades representadas pelos atributos ações_funcionalidades_afetadas e parte_afetada_objeto permitem identificar relações entre modificações no código-fonte e chamados. Determinada modificação em uma funcionalidade pode afetar indiretamente outras funcionalidades, inclusive causando falhas. Deste modo, é importante investigar as relações diretas e indiretas entre chamados e modificações no código-fonte. As relações diretas são percebidas pela mesma funcionalidade (ação+objeto) que o chamado e a modificação no código-fonte referenciam. Boa parte das relações indiretas entre funcionalidades podem ser identificadas através das relações entre objetos do domínio. Por exemplo, considerando-se que um RDF contenha uma paralisação, uma falha no incluir RDF pode estar relacionado com modificações na consulta, inclusão, edição e exclusão de paralização. Para viabilizar a análise de relações indiretas entre funcionalidades, os principais **objetos e relacionamentos do SAO** foram identificados com base nas suas regras de negócio e no seu modelo de dados. Os dados coletados foram armazenados em um banco de dados MySQL para facilitar análise.

A análise dos dados seguiu o algoritmo apresentado na Figura 2. Sua execução foi parte automática e parte manual, pois o conjunto de verificações que precisavam ser realizadas era enorme: 66 chamados x 426 modificações = 28.116 comparações. Primeiro executou-se uma análise automática, de acordo com o algoritmo da Figura 3, para identificar modificações de código relacionadas, direta ou indiretamente, com objetos manipulados nas funcionalidades de interesse no estudo (Tabela 1). Isso permitiu reduzir o espoco da análise manual feita em seguida, conforme o algoritmo da Figura 3. A análise manual considerou não apenas os objetos manipulados, mas também as ações realizadas sobre eles para realizar um filtro nas modificações de código que precisavam ser verificadas. A última verificação manual teve por objetivo verificar se a modificação no código-fonte pode ou não ter causado a falha relatada no chamado.

```
Escolha uma funcionalidade de interesse
Obtenha as modificações relacionadas direta e indiretamente
pelo objeto da funcionalidade de interesse
(executar o algoritmo que identifica modificações de código por objetos)
Filtrar as modificações encontradas pelas ações das funcionalidades relacionadas
(executar o algoritmo que identifica modificações pelas ações das funcionalidades)
Leia cada modificação restante
    Verifique no arquivo PDF que compara os códigos se a modificação tem
    relação com o chamado
    Se houver relação confirmada, escreva o código da modificação, a parte do código-
    fonte modificada e o nome do arquivo do código-fonte
Fim do Leia
```

Figura 2. Algoritmo para identificar modificações de código-fonte relacionadas com chamados.

```
Para cada chamado da macrofuncionalidade de interesse faça
    Imprima o número do chamado, sua descrição, data e objeto da funcionalidade
    Para cada modificação no código-fonte faça
        // identifique as modificações diretas
        Se a data de modificação do código for menor que a data do chamado e o
        objeto afetado no código for igual ao objeto referenciado no chamado
        Então escreva o número da modificação, seu tipo, a parte do código
        afetada, a ação da funcionalidade, a parte do objeto afetada, o pacote e
        a data da modificação
        // Identifique as modificações indiretas
        Se a data de modificação do código for menor que a data do chamado e o
        objeto afetado no código for igual a qualquer objeto relacionado ao objeto
        referenciado no chamado
        Então escreva o número da modificação, o tipo da modificação, a parte
        do código afetada, a ação da funcionalidade, a parte do objeto afetada,
        o pacote e a data da modificação
    Fim do para
Fim do para
```

Figura 3. Algoritmo para identificar modificações de código por objetos.

```
Para cada chamado da funcionalidade de interesse faça
    Imprima o número do chamado, sua descrição, data e objeto da funcionalidade
    Para cada modificação no código-fonte faça
        // modificações diretas
        Leia cada modificação direta no resultado (arquivo) anterior e faça
            Para cada ação da funcionalidade afetada na modificação lida que estiver
            relacionada com a ação da funcionalidade do chamado faça
                Escreva o número da modificação, seu tipo, a parte do código
                afetada, a ação da funcionalidade, a parte do objeto afetada,
                o pacote e a data da modificação
            Fim do para
        Fim do leia
        // modificações indiretas
        Leia cada modificação indireta no resultado (arquivo) anterior e faça
            Para cada ação da funcionalidade afetada na modificação lida que estiver
            relacionada com a ação da funcionalidade do chamado faça
                Escreva o número da modificação, seu tipo, a parte do código
                afetada, a ação da funcionalidade, a parte do objeto afetada,
                o pacote e a data da modificação
            Fim do para
        Fim do leia
Fim do para
```

Figura 4. Algoritmo para identificar modificações de código por ações.

Para cada uma das 18 funcionalidades de interesse nesta etapa do estudo (Tabela 1), a análise automática obteve as modificações relacionadas direta e indiretamente através do objeto da funcionalidade, conforme algoritmo na Figura 3. Ela foi realizada pela *store procedure* do MySQL descrita em [Souza, 2016]. Este resultado foi salvo em dois arquivos texto com conteúdo semelhante ao da Figura 5, um para as modificações relacionadas diretamente e outro para as modificações relacionadas indiretamente.

```
+-------------------------------------------------------------------------------+
| *** Chamado : 214 - Objeto sofre Ação: rdf                                    |
+-------------------------------------------------------------------------------+

+--------+----------+-------------+------------+----------------+-------------------+----------------+---------------------+
| numero | mudanca  | o_que_mudou | quantidade | funcionalidade | objeto_afetado    | pacote         | data_codigo         |
+--------+----------+-------------+------------+----------------+-------------------+----------------+---------------------+
|     10 |  editar  |  interface  |          1 |  consultar     |  qco              |  visao         | 2013-12-31 00:00:00 |
|     11 |  incluir |  interface  |          1 |  responder     |  qco              |  visao         | 2013-12-31 00:00:00 |
|     12 |  incluir |  variavel   |         10 |  _             |  qco              |  dominio       | 2013-12-31 00:00:00 |
|     13 |  incluir |  metodo     |          9 |  get           |  qco              |  dominio       | 2013-12-31 00:00:00 |
|     14 |  incluir |  metodo     |          8 |  set           |  qco              |  dominio       | 2013-12-31 00:00:00 |
|     15 |  editar  |  interface  |          2 |  consultar     |  qco              |  visao         | 2013-12-31 00:00:00 |
|     16 |  incluir |  interface  |          2 |  consultar     |  qco              |  visao         | 2013-12-31 00:00:00 |
|     17 |  incluir |  variavel   |          3 |  _             |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
|     18 |  incluir |  metodo     |          1 |  incluir       |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
|     19 |  editar  |  metodo     |          1 |  cancelar      |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
|     20 |  editar  |  metodo     |          1 |  editar        |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
|     21 |  editar  |  metodo     |          1 |  excluir       |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
|     22 |  editar  |  metodo     |          1 |  consultar     |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
|     23 |  editar  |  metodo     |          1 |  salvar        |  qco              |  apresentacao  | 2013-12-31 00:00:00 |
```

Figura 5. Modificações relacionadas por objetos.

Em seguida, a análise manual reduziu o conjunto de modificações a ser verificado, restringindo-o apenas a modificações com ações de funcionalidades relacionadas, conforme algoritmo na Figura 4. Esta análise das ações precisou ser manual, pois era difícil perceber a priori quais ações referenciadas nos chamados

estariam relacionadas a quais outras. Este resultado foi salvo em três arquivos texto, um para cada macrofuncionalidade de interesse nesta etapa do estudo de caso (Tabela 1), com conteúdo semelhante ao da Figura 6.

```
Chamado 214 -  salvar rdf e incluir, editar e excluir de equipamento, maodeobra, paralisacao, atividade, sms, qco

**** Direto

-|     37 |  editar  |  metodo     |          1 |  salvar       |  rdf           |  apresentacao | 2013-12-31 00:00:00 |
+|    174 |  editar  |  metodo     |          1 |  salvar       |  rdf           |  apresentacao | 2014-05-31 00:00:00 |
-|    216 |  editar  |  metodo     |          3 |  salvar       |  rdf           |  apresentacao | 2014-09-30 00:00:00 |
+|    240 |  editar  |  metodo     |          1 |  salvar       |  rdf           |  servico      | 2014-10-31 00:00:00 |


**** Primeiro Grau

-|     18 |  incluir |  metodo     |          1 |  incluir      |  qco               |  apresentacao   | 2013-12-31 00:00:00 |
+|     20 |  editar  |  metodo     |          1 |  editar       |  qco               |  apresentacao   | 2013-12-31 00:00:00 |
+|     21 |  editar  |  metodo     |          1 |  excluir      |  qco               |  apresentacao   | 2013-12-31 00:00:00 |
+|     23 |  editar  |  metodo     |          1 |  salvar       |  qco               |  apresentacao   | 2013-12-31 00:00:00 |
+|     24 |  editar  |  metodo     |          1 |  validar      |  qco               |  apresentacao   | 2013-12-31 00:00:00 |
+|     45 |  editar  |  metodo     |          1 |  salvar       |  qco               |  apresentacao   | 2014-01-31 00:00:00 |
-|     86 |  incluir |  interface  |          1 |  incluir      |  maodeobra         |  visao          | 2014-02-28 00:00:00 |
-|     88 |  editar  |  metodo     |          1 |  incluir      |  maodeobra         |  infraestrutura | 2014-02-28 00:00:00 |
```

Figura 6. Modificações relacionadas por objetos e por ações.

Por fim, com um número ainda mais reduzido de modificações a serem verificadas, a análise manual passou a analisar as comparações do código fonte nos arquivos PDFs com os chamados, conforme o último bloco do algoritmo na Figura 2. Para cada chamado, inspecionou-se a comparação do código-fonte em PDF de cada modificação que ainda poderia estar relacionada a ele. O objetivo desta análise manual qualitativa foi verificar se o significado das alterações no código-fonte poderia ter relação conceitual direta ou indireta com os problemas relatados nos chamados. O conhecimento do domínio e a experiência de um dos pesquisadores no processo de desenvolvimento do SAO contribuíram bastante para esta análise. Quando se confirmava uma possível relação entre uma modificação no código-fonte e um chamado, os pesquisadores escreveram num arquivo texto o código da modificação relacionada, a parte do código-fonte afetada (geralmente um método) e o nome do arquivo fonte, com resultado similar à Figura 7 para o chamado 214.

```
**** Análise do Código
20 - edição no método alterar() do QCOBean
21 - Edição no método excluir() do QCOBean
23 - Edição no método salvarQCO() do QcoBean
24 - Edição no método validarQcoEntidadeComErrosAntesSalvar()  do QCOBean
45 - Edição no método salvarQCO() do QcoBean
150 - Edição adicionarParalisacaoAListA() do RelatoriodeFrenteBean
162 e 373 - Edição validarHorario() do TurnoBean
```

Figura 7. Modificações com código relacionado ao chamado 214.

5. Resultados

Na investigação por relações entre modificações no código-fonte e chamados, dois grupos disjuntos de modificações foram identificados: modificações confirmadas e não confirmadas pela inspeção do código-fonte. As primeiras representam (resultados positivos) possíveis origens das falhas referenciadas nos chamados (resultado final do algoritmo na Figura 2). As segundas representam (resultados falso-positivos) modificações no código-fonte relacionadas com as funcionalidades dos chamados que não geraram as falhas reportadas (resultado parcial do algoritmo na primeira parte da Figura 2, resultante da execução dos algoritmos da Figura 3 e Figura 4). Além disso, essas modificações também foram classificadas pelo relacionamento direto ou indireto com as funcionalidades referenciadas. Modificações diretas afetam a própria funcionalidade referenciada no chamado. Já as indiretas afetam outras funcionalidades

relacionadas pelos objetos e ações do domínio. A Tabela 2 apresenta a quantidade mensal de modificações relacionadas com chamados das macrofuncionalidades "Manter RDF" (esq), "Manter QCO" (centro) e "Gerar Relatório Detalhado do RDO" (dir.). As quantidades estão estratificadas pela confirmação no código-fonte, e pela relação direta ou indireta com a funcionalidade.

Tabela 2. Quantidade de modificações no código relacionadas com os chamados.

	Manter RDF				**Manter QCO**				**Gerar Relatório Detalhado do RDO**			
	confirmadas pelo código		**relacio. pela funciona-lidade**		**confirmadas pelo código**		**relacio. pela funciona-lidade**		**confirmadas pelo código**		**relacio. pela funciona-lidade**	
	dir.	ind.	dir.	ind.	dir.	ind.	dir.	ind.	dir.	ind.	dir.	ind.
dez/13	6	4	29	15	11		21	2	18		70	
jan/14	11	1	14		12		7		11		101	
fev/14	3	15	87	127	3		18		8		65	
mar/14		3		7			4		4		26	
abr/14	5	15	14	111	1		9		7		41	
mai/14	4	2	1	17	12		6		1		43	
jun/14		1	1	3	6				2		34	
jul/14		8		33	3							
ago/14	9	8	20	8	3							
set/14	7	4	41	8	10				2		6	
out/14	4		5	14								
nov/14											1	
dez/14	1	1					1					
jan/15												
fev/15					1							
mar/15												
total	50	62	212	343	62	0	66	2	53	0	387	0

O código-fonte do SAO sofreu 50 modificações diretamente relacionadas com "Manter RDF" ao longo do período analisado, bem como 62 modificações indiretas em outras funcionalidades que afetam "Manter RDF". Além dessas modificações confirmadas na inspeção do código-fonte, outras 212 modificações diretas e 343 indiretas estão relacionadas com "Manter RDF", mas não se confirmaram como possíveis origens de falhas. Em todos os casos, as modificações ocorreram durante pelo menos 9 meses. O código-fonte do SAO sofreu 62 modificações diretamente relacionadas com falhas reportadas para "Manter QCO" ao longo de 10 meses durante o período analisado, mas nenhuma modificação indireta em outras funcionalidades que a afetam. Também ocorreram outras 66 modificações diretas e 2 indiretas relacionadas com "Manter QCO", que não se confirmaram como possíveis origens de falhas. Estas foram dispersas em até 9 meses. O código-fonte do SAO sofreu 53 modificações diretamente relacionadas com falhas reportadas sobre "Gerar Relatório Detalhado do RDO" ao longo de 8 meses, mas nenhuma modificação indireta em outras funcionalidades que a afetam. Ocorreram ainda 387 modificações diretas e nenhuma indireta que não geraram as falhas reportadas para esta funcionalidade.

Cada chamado está relacionado com quantas dessas modificações? As Figuras 8, 9 e 10 indicam a quantidade mensal de modificações relacionadas com "Manter RDF", "Manter QCO" e "Relatório Detalhado do RDO", respectivamente.

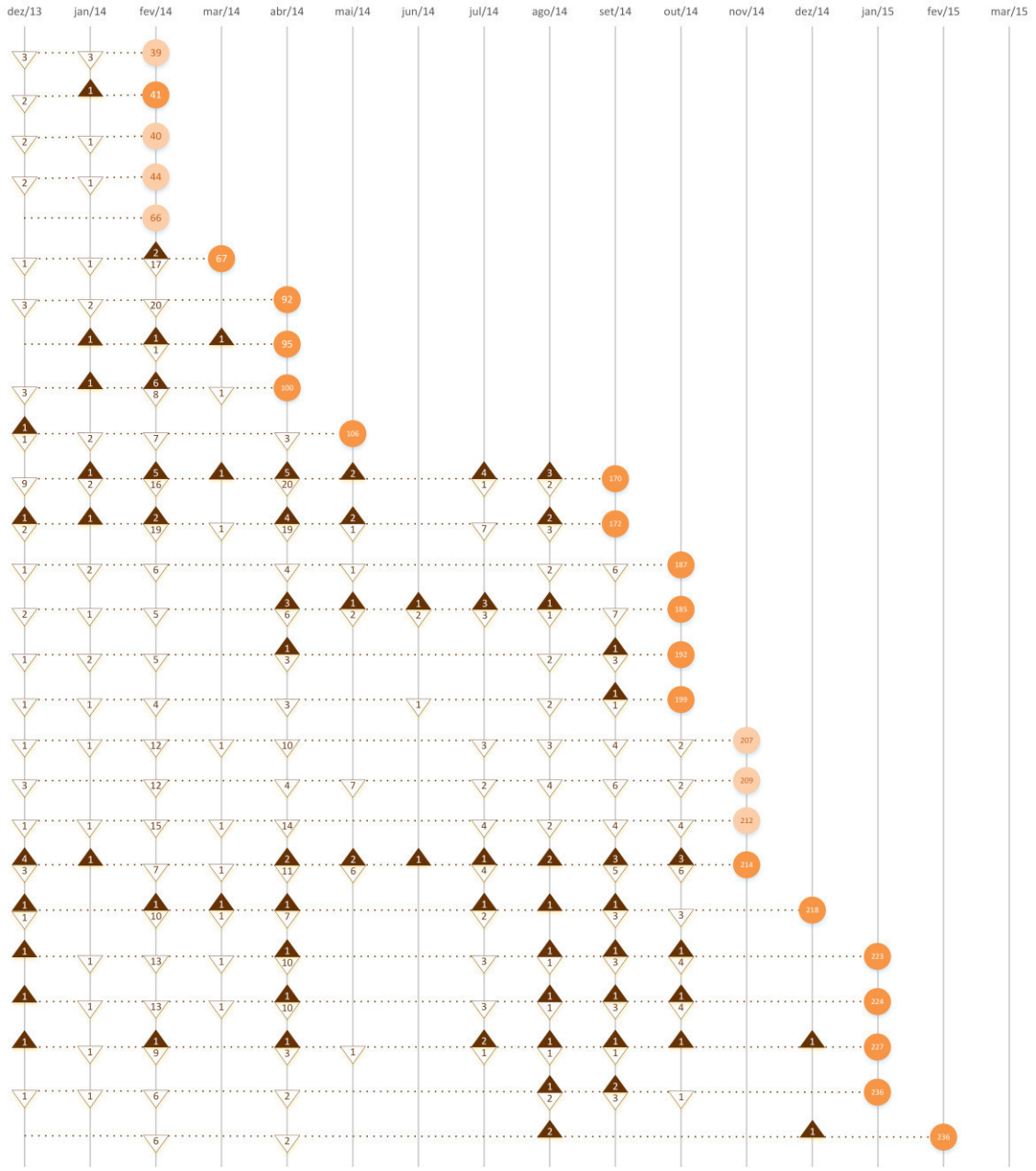

Figura 8. Modificações relacionadas por código ou por funcionalidade para cada chamado de "Manter RDF".

A quantidade de modificações foi representada por número dentro de um triângulo. Um triângulo marrom apontado para cima representa as modificações que foram confirmadas como possíveis origens das falhas reportadas nos chamados (resultado positivo). O triângulo branco apontado para baixo representa modificações relacionadas com a funcionalidade do chamado, mas que não representam origem da falha reportada (resultado falso-positivo). Os chamados estão representados por um

círculo laranja. Os chamados que não tiveram origem identificada nas modificações no código-fonte foram representados com um tom de laranja mais claro.

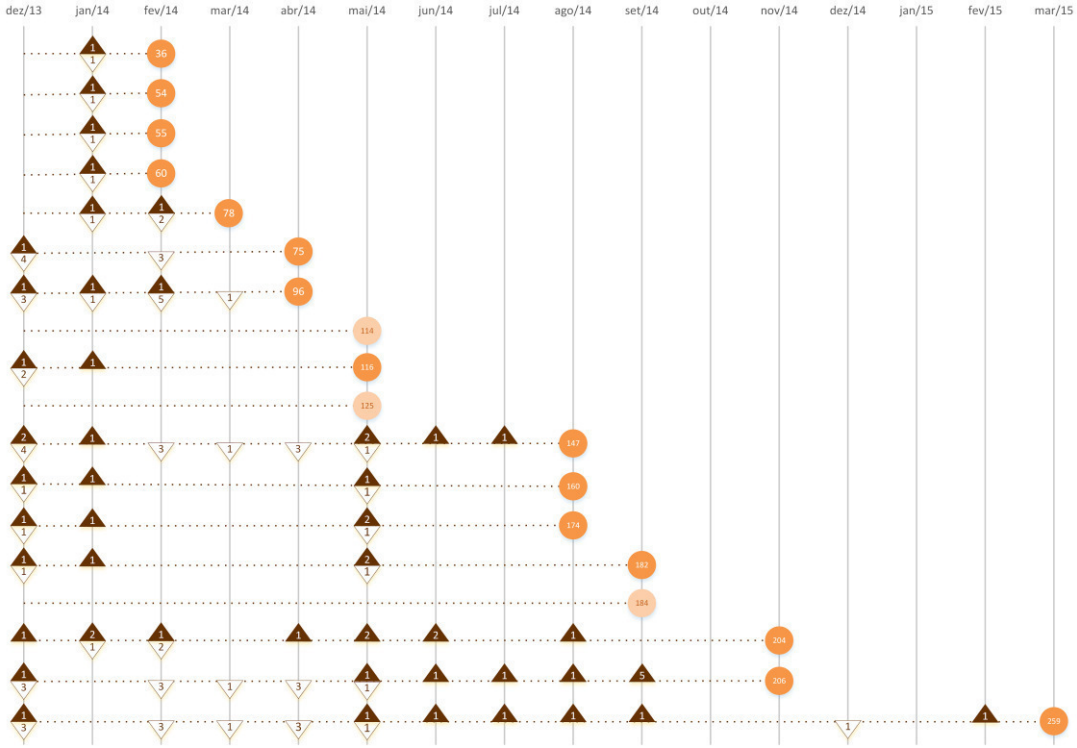

Figura 9 - Modificações relacionadas por código ou por funcionalidade para cada chamado de "Manter QCO".

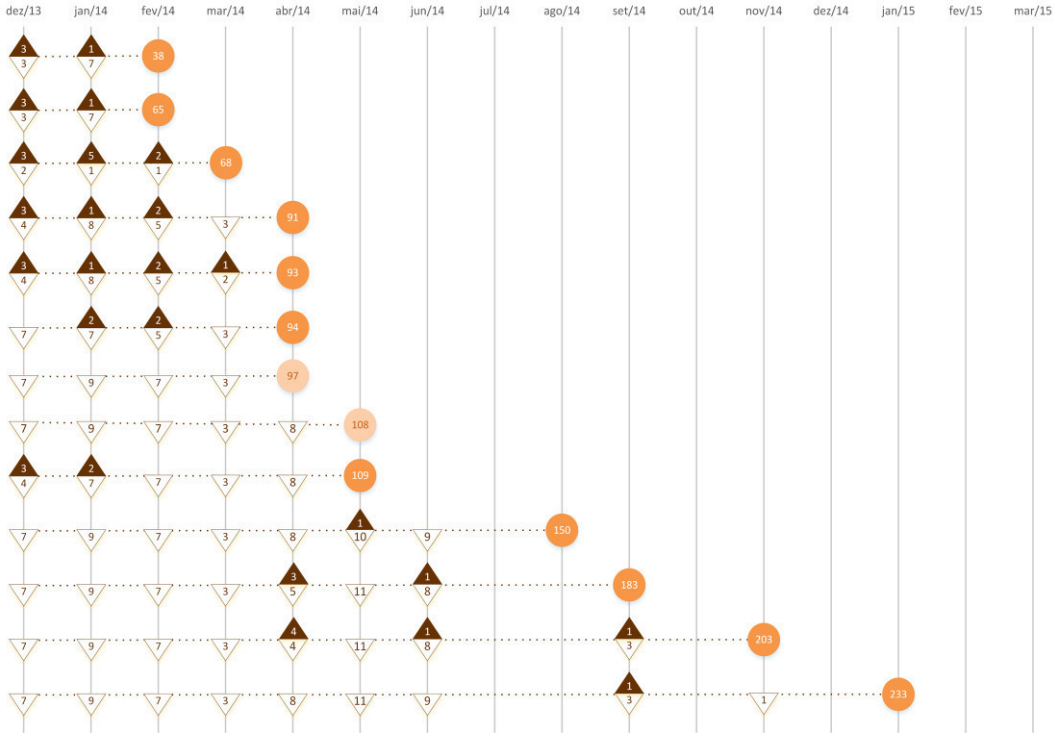

Figura 10. Modificações relacionadas por código ou por funcionalidade para cada chamado de "Relatório Detalhado do RDO".

Observamos que 21% dos chamados não possuem modificações confirmadas no código: 7 em "Manter RDF", 3 em "Manter QCO" e 2 em "Relatório Detalhado do RDO". As falhas reportadas em muitos chamados podem ter origem nas modificações de código que ocorreram em vários meses, chegando a 9 meses de modificações recorrentes. Algumas das modificações inclusive estão relacionadas com chamados que ocorreram muitos meses depois, chegando a mais de um ano de intervalo entre a modificação (possível origem da falha) e a ocorrência do chamado (descoberta da falha).

Figura 11. Modificações relacionadas por código para cada chamado de "Manter RDF".

Será que uma modificação pode ter causado falhas em mais de um chamado? As Figuras 11, 12 e 13 apresentam as relações de cada modificação que pode ter sido a origem de falhas reportadas em cada chamado de "Manter RDF", "Manter QCO" e "Relatório Detalhado do RDO", respectivamente. Nestas figuras a modificação que pode ter gerado a falha foi representada por um hexágono marrom. Os chamados continuam representados por círculos laranja; mais escuro para chamados com modificações que podem ter gerado a falha reportada e mais claro para chamados sem modificações que geraram a falha reportada. Os relacionamentos estão representados por linhas marrons que ligam modificações e chamados. Nessas figuras é possível perceber que boa parte das modificações afeta mais de um chamado.

Figura 12 - Modificações relacionadas por código para cada chamado de "Manter QCO".

Em média, uma modificação está relacionada com 2,13 chamados, com desvio padrão de 1,55 chamados. Dezessete modificações (16%) têm relação com de 4 até 7 chamados. Trinta e quatro modificações (33%) se relacionam com 2 ou 3 chamados. Cinquenta e três modificações (51%) se relacionam com apenas um chamado.

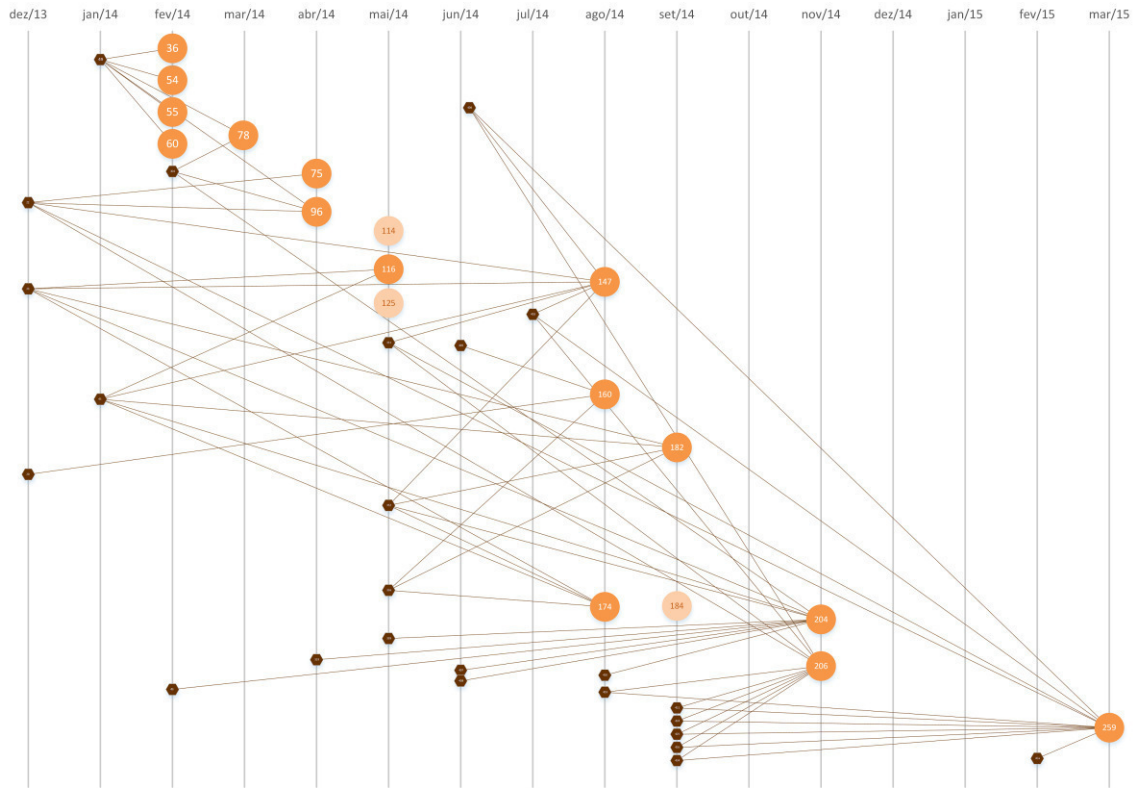

Figura 13. Modificações relacionadas por código para cada chamado de "Relatório Detalhado do RDO".

6. Discussão

Doze chamados (21%) associados às três macrofuncionalidades analisadas não puderam ser relacionados com nenhuma modificação do código-fonte após o lançamento do SAO. Estes problemas podem ser de infraestrutura (banco de dados, permissão de acesso, etc.) ou tiveram origem no código-fonte da primeira versão. Os outros 45 chamados podem ter sido causados por falhas durante desenvolvimento ou manutenção, já que existem modificações relacionadas e boa parte delas é recorrente. Mesmo depois de quase um ano, surgiram chamados associados a modificações de código realizadas na manutenção. Quase metade das modificações se relaciona com mais de um chamado.

O que poderia ter sido feito durante o processo de desenvolvimento do SAO para evitar que o usuário passasse por tantos problemas? Certamente a atividade de teste [Bourque e Fairley, 2014] poderia ter contribuído bastante para a qualidade do SAO antes dele ser disponibilizado em produção. Vários tipos de teste poderiam ser melhorados e realizados durante o processo de desenvolvimento do SAO: testes unitários, de integração, de sistema, de aceitação e de regressão [Delamaro et al., 2007].

Testes unitários e **de integração** provavelmente identificariam falhas causadas por modificações diretas no código-fonte, como as 165 indicadas na Tabela 2. **Testes de integração** e **de sistema** provavelmente identificariam falhas causadas por modificações indiretas no código, como as 62 modificações encontradas. O que o desenvolvedor faria diante das mudanças no código-fonte que não geraram falhas reportadas nos chamados? Como o desenvolvedor poderia diferenciá-las daquelas que realmente geraram falhas?

Sem um minucioso trabalho de inspeção manual de código, o desenvolvedor pode ter dificuldade de diferenciar possíveis modificações que afetam ou não as funcionalidades. Nestes casos, ou se faz uma boa rastreabilidade e bom controle de modificações para análise de impacto, ou se realiza um volume maior de **testes de regressão** para cobrir funcionalidades relacionadas.

A equipe de desenvolvimento do SAO priorizou testes manuais em detrimento de testes automatizados, mesmo durante a manutenção do sistema. Todavia, é impraticável executar muitos testes de regressão manualmente com regularidade, pelo tempo e esforço necessários. A opção viável seria aumentar a cobertura e melhorar os testes automatizados para execução regular de testes de regressão. Estes testes provavelmente encontrariam boa parte das falhas que surgiram durante modificações de código. Se as falhas fossem identificadas e corrigidas rapidamente, não teríamos uma quantidade e recorrência tão grande de chamados de correção no SAO. Sem execução sistemática de testes de regressão durante a manutenção, é grande o risco de entregar correções de falhas que geram outras falhas inadvertidamente.

7. Considerações Finais

O Sistema de Acompanhamento de Obras (SAO) tem sido desenvolvido em uma empresa de energia no Brasil. Como os usuários enfrentaram muitos problemas após o lançamento, o sistema, que poderia representar uma vantagem para a empresa, tornou-se desvantagem por falta de qualidade, principalmente para reputação da equipe de TI.

Este artigo apresentou a última etapa de um estudo de caso do SAO que investigou as relações entre chamados de correção e modificações do código-fonte durante a manutenção do sistema, entre dezembro de 2012 até março de 2015. O estudo concentrou-se nas três macrofuncionalidade que receberam maior número de chamados neste período, pois foi analisar manualmente todas as macrofuncionalidades era inviável. Percebemos uma boa quantidade de modificações no código relacionada a chamados. A maioria dos chamados (79%) pode ser relacionada com modificações no código-fonte após o lançamento do sistema, com uma média de quase 4 modificações por chamado. Isso reforça a necessidade de manter ativa a preocupação com a qualidade durante a manutenção do sistema, ainda que as modificações sejam para correção de falhas como no caso do SAO. As modificações que afetaram diretamente as funcionalidades referenciadas nos chamados ressaltaram a importância de realizar testes unitários e de integração. As modificações que afetaram indiretamente as funcionalidades referenciadas nos chamados destacaram a relevância de testes de integração, de sistema e, principalmente, os de regressão. Como o volume de testes necessários costuma ser grande, é fortemente recomendado realizar testes automatizados para viabilizar uma boa cobertura e frequência de execução ao longo de todo o desenvolvimento e manutenção do sistema.

A equipe responsável pelo desenvolvimento do SAO ainda não compreendia muito bem as consequências de não cuidar adequadamente dos testes durante a manutenção do sistema, além de verificar se as correções funcionam. A credibilidade perante os usuários e empresa, bem como a indisponibilidade da equipe para realizar atividades diferentes de correção de falhas foi um custo alto para este descuido. Apesar de a atividade de testes estar bem estabelecida e divulgada na academia [Craig e Jaskiel,

2002; Delamaro et al., 2007], ela ainda precisa ser melhorada na prática de desenvolvedores na indústria de software brasileira.

A literatura relata várias pesquisas sobre fatores que influenciam a qualidade de sistemas. Geralmente as pesquisas se concentram em aspectos mais técnicos, como a identificação de falhas comuns durante a codificação [Lu et al., 2008], a predição de falhas [D'ambros et al., 2010] e a eficácia do desenvolvimento dirigido por testes [Filho et al., 2012]. O propósito desta pesquisa é ligeiramente diferente, pois endereça fatores humanos fornecendo subsídios para **conscientizar profissionais da indústria**. Esperamos que a experiência negativa do SAO no período analisado possa contribuir para os profissionais **compreenderem a importância** de cuidar adequadamente da atividade de testes durante todo o processo de desenvolvimento e manutenção de um sistema. Os dados reais aqui apresentados podem auxiliar desenvolvedores a refletirem sobre sua prática profissional e identificar possíveis descuidos, de modo a motivar seu aprimoramento. A atividade de testes precisa deixar de ser considerada como apenas um assunto acadêmico para ser incluída também como prática cotidiana na indústria.

Referências

Grubb, P; Takang, A.A. (2003) **Software Maintenance: Concepts and Practice**. 2nd ed., World Scientific Publishing, 2003.

Bourque, P.; Fairley, R.E. (2014) **Guide to the Software Engineering Body of Knowledge (SWEBOK)**. IEEE Computer Society Press.

Craig, R. D.; Jaskiel, S. P. (2002) **Systematic Software Testing**. Artech House Publishers.

Delamaro, M.E.; Maldonado, J.C.; Jino, M. (2007) **Introdução ao Teste de Software**, Elsevier.

White, R.W. e Roth, R.A. (2009) **Exploratory Search: Beyond the Query-Response Paradigm**, Morgan and Claypool.

Stake, R.E. (1995) **The Art of Case Study Research**. Sage Publications.

Sousa, L.A.C. (2016) **Estudo Exploratório da Atividade de Testes num Sistema de Acompanhamento de Obras**. Dissertação de Mestrado Profissional em Engenharia de Software. Universidade Federal do Rio Grande do Norte.

Lu, S.; Park, S.; Seo, E.; Zhou, Y. (2008) Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In **Proceedings of the 13th international conference on Architectural support for programming languages and operating systems** (ASPLOS XIII).

D'Ambros, M.; Lanza, M.; Robbes, R. (2010) An Extensive Comparison of Bug Prediction Approaches. **In Proceedings of 7th IEEE Working Conference on Mining Software Repositories** (MSR 2010).

Filho, M.C.; Vasconcelo, J.L.; Santos, W.B.; Silva, I.F. (2012) Um Estudo de Caso sobre o Aumento de Qualidade de Software em Projetos de Sistemas de Informação que Utilizam Test Driven Development. **Anais do VIII Simpósio Brasileiro de Sistemas de Informação** (SBSI 2012).