Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Análise da Evolução de Software com Séries Temporais

Nelson Baptista da Fonte (nº 29475)

2º Semestre de 2009/10

28 de Julho de 2010

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Análise da Evolução de Software com Séries Temporais

Nelson Baptista da Fonte (nº 29475)

Orientador: Prof. Doutor Miguel Carlos Pacheco Afonso Goulão

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2009/10
28 de Julho de 2010

À minha família.

Agradecimentos

Gostaria de agradecer e expressar a minha sincera gratidão e reconhecimento ao meu orientador Miguel Carlos Pacheco Afonso Goulão, PhD, pelo seu encorajamento, apoio e orientação permanente na elaboração e conclusão desta dissertação. Durante este trabalho, ele foi uma inesgotável fonte de conhecimento. Destacando os seus méritos não só é apropriado e justo, mas ainda uma expressão da minha gratidão pelo seu empenho e profissionalismo.

Os meus agradecimentos vão também para:

- A minha família, principalmente aos meus pais Heitor e Maria José, e ao meu irmão Mickaël, por todo o esforço nestes anos para que nunca me faltasse nada.
- A minha namorada Sónia que me aturou sempre que foi preciso e me apoiou incondicionalmente durante a realização desta dissertação.
- Ao Paulo Severino um grande amigo de longa data que me reviu a tese muito encima da hora, mas que fez um excelente trabalho apesar de não ser desta área.
- Aos meus amigos e colegas (sem importância na ordem), na primeira fase do meu percurso académico: Hélio Pereira, Luís Oliveira, Marco Gonçalves, Pedro Correia, Pedro Chambel, Rui Bom Garcia, Hélio Dolores, Rúben Jorge, Francisco Costa pelos excelentes momentos passados em convívio, sempre com uma boa disposição e um grande espírito de camaradagem e companheirismo, mesmo nas horas de estudo intensivo. Na segunda fase do meu percurso académico, têm de ser mencionados: Luís Silva, Roberto Félix, Tiago Amorim, Pedro Gabriel e Hugo Aguiar, por um mestrado repleto de bons momentos.
- A todos os professores e colegas que partilharam comigo os corredores e as salas da FCT/UNL nestes anos.

A todos,

Muito Obrigado!

Resumo

Um sistema de software nunca está terminado. Mesmo depois de ter sido entregue, o software continua a evoluir. Por esta razão, governos, empresas, e comunidades *open source* gastam muitos recursos regularmente para corrigir, adaptar, ou melhorar, os seus sistemas de software. Alguns estudos referem que cerca de 90% dos recursos das empresas dedicados ao software são gastos em actividades de manutenção. Isso implica que apenas 10% é dedicado para outras actividades, entre as quais o desenvolvimento de novos projectos. Isto representa uma oportunidade para, com um melhor planeamento, se tornar o processo de software mais eficiente, com importantes ganhos económicos que daí resultam. É por isso que a capacidade de desenvolver software de uma forma rápida e fiável é um grande desafio na Engenharia de Software. Uma possível técnica para ajudar a reduzir custos e produzir um software de qualidade é, a previsão do seu comportamento no futuro. Para os gestores de projecto e programadores, prever a evolução do software será de grande utilidade, pois permitirá direccionar esforços para partes que necessitem uma maior intervenção. Para a previsão ser possível, é necessário analisar a história da vida do software, que está normalmente guardada nos repositórios de dados dos projectos.

Pretendemos por isso, efectuar uma análise da evolução do IDE Eclipse, usando séries temporais. Esta análise permitirá visualizar a evolução do número de defeitos do Eclipse ao longo do tempo e sua previsão no futuro. Serão usados dados do sistema de rastreio de defeitos. Isto permitirá identificar possíveis padrões e tendências na distribuição do número de defeitos, permitindo a criação de um modelo fiável de previsão. O resultado desta dissertação constituirá mais um caso de estudo da evolução de um sistema de sucesso, duradouro e bastante usado, que diverge nos objectivos de trabalhos anteriores sobre o Eclipse, mas que com outro estudo é um reforço da utilização da análise de séries temporais, uma técnica insuficientemente explorada, no contexto do estudo da evolução de software, particularmente na previsão dessa evolução.

Palavras-chave: Evolução de Software, Previsão, Séries Temporais, Engenharia de Software, Manutenção, Eclipse, Open Source

Abstract

A software system is never finished. Even after it was delivered, the software continues to evolve. For this reason, governments, businesses, open source communities and spend a lot of resources regularly to correct, improve or adapt their software systems. Earlier research states that up to 90% of of the resources of companies dedicated to software, are spent on maintenance activities. This indicates that only 10% is devoted to other activities including the development of new projects. This represents an opportunity, with better planning, to make the software process more efficient, which should yield significant economic benefits. The ability to develop software quickly and reliably is a major challenge in Software Engineering. One possible technique to help reducing costs without sacrificing software quality is the forecast of their behavior in the future. For project managers and developers to predict the evolution of the software will be useful, because it will help directing efforts for parts that require more intervention. Analysing the life history of the software, which is normally stored in the repositories of the projects or, in some cases, data repositories.

We intend therefore to analyse the evolution of the Eclipse IDE, using time series. This analysis will display the evolution of the number of defects on the Eclipse over time and its prediction in the future. Will use data from the defects tracking system. This will help identify possible patterns and trends in the distribution of the number of defects, enabling the creation of a reliable model for forecasting. The result of this work will be another case study of the evolution of a system of success, durable and widely used, which differ in the objectives of earlier work on Eclipse, but with another study is an strengthening the use of time series analysis, a technique not sufficiently explored in the context of studying the evolution of software, particularly in forecasting the evolution of Eclipse.

Keywords: Software Evolution, Forecasting, Time Series, Software Engineering, Maintenance, Eclipse, Open Source

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

A Evolução do Software é um tema de investigação muito activo da Engenharia de Software dado que o software se tornou omnipresente e indispensável numa sociedade cada vez mais informatizada. Além disso, ou como efeito colateral, o software tem evoluído cada vez mais. Assim, embora possamos ser capazes de resolver alguns dos problemas relacionados com a evolução do software, iremos sempre continuar a encontrar novos problemas que precisam de ser resolvidos.

1.1 Introdução geral ou Motivação

A evolução de software é um domínio de investigação crucial, complexo e muito importante na Engenharia de Software. Tem sido o tópico de numerosas conferências internacionais, *workshops*, livros e publicações científicas. Em contrapartida, representa um problema recorrente para as organizações e necessita de soluções práticas e escaláveis com a finalidade de garantir a confiança, qualidade e fiabilidade. Esta situação têm-se tornado ainda mais crucial e crítica em domínios aplicacionais onde o software se encontra distribuído geograficamente e que envolvem múltiplas entidades com interesses no software (gestores, desenhadores, implementadores, clientes, ...) onde os recursos e requisitos têm de ser conciliados. É um processo inevitável, difícil de prever e dispendioso. Se, pelo contrário, for bem concretizada, é um factor

de sucesso para a inovação do negócio onde se insere, essencial à competitividade e sobrevivências das empresas. Essas mudanças ocorrerão sempre. Os requisitos mudam devido a novos clientes, novas exigências de actuais clientes, mudanças na organização estrutural, novos competidores ou mesmo mudanças da legislação. Podem também ocorrer devido ao ciclo *feedback* onde o próprio software modificado é a razão da alteração do ambiente que por sua vez vai originar novos requisitos para o software, a prazo. As mudanças representam também um desafio para uma melhoria contínua da qualidade de software, já que envolvem a correcção de defeitos. A evolução de software tem como finalidade melhorar a qualidade, performance ou fiabilidade, retardando, na medida do possível, o envelhecimento/erosão do software que eventualmente acontecerá.

Para isso é necessário apoiar/dar suporte à evolução de software em vários níveis de abstracção, em paradigmas emergentes e na associação de variadas tecnologias (orientada a aspectos, orientada a serviços, entre outras) nos mais relevantes domínios aplicacionais (banca, finanças, redes *wireless*, entre outros).

A Evolução do Software é uma actividade de investigação transversal que é necessária em todas as actividades da Engenharia de Software (especificação de requisitos, análise e desenho, programação e desenvolvimento, construção de baterias de ensaio de software), em todos os níveis de abstracção (por exemplo, código executável, código fonte, modelos de desenho), em todos os desenvolvimentos de paradigmas de software, para todas as tecnologias numa grande variedade de domínios aplicacionais e para muitos tipos de entidades diferentes interessadas no software.

1.2 Descrição e contexto

Numerosos estudos científicos sobre grandes sistemas de software têm demonstrado que grande parte do esforço realizado pelos elementos de desenvolvimento e o custo despendido em grandes projectos de software revertem para a manutenção e evolução dos sistemas de software existente ao contrário do desenvolvimento de um novo projecto a partir do zero [10, 23, 78]. Isto deve-se principalmente à necessidade de evolução progressiva do software dos sistemas para poder lidar com os requisitos que vão mudando.

Na tentativa de reduzir o esforço e custos das operações de manutenção e evolução de sistemas existentes, muitos investigadores têm sugerido, boas práticas, orientações, técnicas e ferramentas que analisam o software e localizam onde estão os problemas actuais do software.

No entanto, poucos se têm dedicado a tentar prever como é que o software vai evoluir no futuro e onde será necessária maior intervenção das operações de manutenção e evolução. Essa capacidade de prever, se alcançada, permitiria a redução do esforço e custo, já que possibilitaria a tomada de decisões adequadas por parte da equipa de desenvolvimento, facilitando o redireccionamento de recursos para as partes ou componentes mais críticas, ou mesmo tentando definir formas de solucionar o problema, para que o previsto não aconteça como esperado.

1.3 Proposta apresentada

O trabalho proposto pretende fazer um estudo sobre a evolução de um software e a consequente análise que poderá permitir a previsão do seu comportamento no futuro. Será adicionalmente tentado a adição de informação proveniente dos sistema de rastreio para enriquecer a informação recolhida dos repositórios. A previsão será efectuada com base em séries temporais propícias a previsão, previsão essa que é um objectivo difícil de alcançar na área da Engenharia de Software.

Este trabalho incide, a exemplo de estudos realizados por Wermelinger [81] e Mens [59], sobre o Eclipse. Os autores efectuam um estudo sobre o mesmo software proposto para esta dissertação. Apesar desta dissertação coincidir com os estudos acima citados no objecto de estudo, difere desses trabalhos nos seguintes aspectos: (1) nos trabalhos de Wermelinger e Mens, apenas se tira partido do repositório do código fonte do Eclipse, enquanto que nesta dissertação, será utilizada a informação proveniente do sistema de rastreio das acções de evolução (Bugzilla); (2) os trabalhos anteriores limitam o seu âmbito à análise de dados históricos, enquanto que nesta dissertação, um dos objectivos é precisamente a proposta de modelos de previsão; (3) efectuamos uma análise a nossa série temporal para identificar padrões e tendências, o que não é feito nos trabalhos de Wermelinger e Mens.

1.4 Principais contribuições previstas

As principais contribuições previstas desta dissertação são:

- reconstrução da história da evolução do Eclipse, com a produção de um pacote experimental reutilizável em futuros estudos.
- análise da evolução do Eclipse.

- proposta de um modelo para previsão de defeitos do Eclipse.
- utilização de séries temporais para a previsão de *lead time* em acções de evolução.

O tópico que se segue, é a principal contribuição que esta dissertação tem em comum com os tópicos de interesse que o grupo dedicado à Evolução de Software do consórcio European Research Consortium for Informatics and Mathematics (ERCIM [1]) procura responder.

- análise da evolução de todos os tipos de artefactos de software a qualquer nível (e.g., relatórios de defeitos, informação de controlo de versão, *log files*, história de versões)

Em [6] reafirma-se a necessidade de mais trabalhos de pesquisa sobre a Evolução de Software, uma área onde continuam a existir desafios por resolver, apesar dos progressos que se fizeram sentir desde da elaboração dos desafios previstos para a área em 2005 [60].

1.5 Organização do Documento

- **Capítulo 1**. Apresentação de uma visão geral do trabalho a ser desenvolvido, nomeadamente a sua motivação, o seu contexto, a descrição do problema, o objectivo da proposta e as suas principais contribuições.
- **Capítulo 2**. Introdução do conceito de Evolução de Software, onde é descrito mais ao pormenor o termo "Evolução" e uma ligação com a Engenharia de Software e apresentamos uma série de conceitos/fundamentos que estão constantemente presentes na vida do software.
- **Capítulo 3**. Discussão das séries temporais, onde são apresentados vários tipos e modelos de séries e os problemas que as mesmas podem resolver.
- **Capítulo 4**. Neste capítulo são discutidas as fontes de informação a usar no nosso trabalho. Estas incluem sistemas de rastreio de defeitos que recolhem a informação relevante para a análise, seguindo-se um estudo dividido em duas partes, modelos que guardam a informação de uma maneira e ferramentas que visualizam a informação dos modelos ou directamente dos repositórios e sistemas. Este capítulo é encerrado com trabalhos sobre a previsão.

[1] http://wiki.ercim.eu/wg/SoftwareEvolution

- **Capítulo 5**. Apresentação do Eclipse, cuja evolução constituirá o caso de estudo a desenvolver durante a elaboração da dissertação.
- **Capítulo 6**. Neste capítulo é apresentado todo o trabalho efectuado sobre a análise e previsão da evolução do Eclipse com séries temporais. Este capítulo está estruturado seguindo as normas de relato de experiências no contexto da Engenharia de Software Experimental, para facilitar a integração dos resultados nele obtidos com os obtidos noutros estudos relacionados.
- **Capítulo 7**. Por último no capítulo 7, são apresentadas as conclusões, o impacto e trabalho futuro desta dissertação.

2

Evolução de Software

Numa sociedade cada vez mais informatizada, o software tem-se tornado indispensável para a sobrevivência de muitas companhias e empresas que lhe são dependentes. Devido a essa dependência, a necessidade de criar software fiável e duradouro é fundamental sendo esse um dos principais objectivos dos engenheiros de software. O software fiável é desde há muito considerado como software sem *bugs* ou defeitos. Este conceito estabelece uma divisão no rumo dos trabalhos de pesquisa. Uns são focados para a prevenção, detecção e reparação de erros nos vários estados do processo de desenvolvimento de software, outros têm como objectivo, o aumento de produtividade que visa a redução de custos de produção, sendo estes bastante importantes para a sobrevivência económica das empresas. A constante pressão no software exercida pelos extremos graus de complexidade e diversidade que a sociedade actual impõem, faz com que este esteja constantemente em mudança. Estas mudanças podem ser consideradas de dois tipos, mudanças que alteram o comportamento, que acarretam a inserção de novas funcionalidades ou mudanças que não alteram o comportamento, tarefas de manutenção.

Este capítulo está organizado em quatro secções. Na secção 2.1 o termo evolução será definido e discutido, de modo a clarificar o seu significado na Engenharia de Software e contrastar essa definição com a usada noutras áreas. Na secção 2.2, uma discussão sobre evolução de software, referindo a especificação ***S**pecification **P**roblem **E**volution*, as leis da evolução e o

princípio da incerteza como partes integrantes da mesma. A secção 2.3 discute a manutenção de software, o seu papel nesta área e, por fim, é feita a comparação entre os conceitos de evolução e manutenção de software.

2.1 Evolução

O termo "Evolução" descreve um fenómeno que está presente nos mais diversos domínios, tais como a biologia, sociologia, cidades, conceitos, teorias, ideias, e que se vai reflectindo ao longo do tempo no seu próprio contexto. O termo evolução [49] reflecte um processo de mudança progressivo nos atributos ou características da entidade que está evoluindo.

Uma entidade ou uma colecção de entidades podem ser ditas a evoluir, se o seu valor ou capacidade se altera ao longo do tempo. Isto significa que individualmente ou colectivamente, as entidades se vão adaptando em função de um ambiente em mudança.

As mudanças são geralmente incrementais e de pequena dimensão comparado com a dimensão da entidade, mas também podem ocorrer concorrentemente em algumas propriedades da entidade. Em áreas como o desenvolvimento de software, muitas mudanças alegadamente independentes podem ser implementadas em paralelo. À medida que as mudanças ocorrem como parte da evolução global, propriedades que não são mais apropriadas podem ser removidas ou simplesmente desaparecer e novas propriedades podem emergir.

A Biologia, área que introduziu o conceito de evolução, estuda a evolução incidente sobre as espécies de seres vivos. Uma possível analogia entre a evolução das espécies e a evolução de software é apresentada por Mens e Demeyer [58] onde é dito que: as espécies correspondem aos modelos de alto nível que normalmente se tenta definir no contexto do desenvolvimento de software. Uma descrição de arquitectura é, assim, a descrição de toda uma espécie de sistemas de software. Se a evolução ocorre no software então podemos esperar que isso seja observável na arquitectura. Novas arquitecturas são criadas como melhoramentos de arquitecturas precedentes, originando espécies evoluídas.

2.2 Fundamentos da Evolução do Software

Desde o início da década de 60, o termo evolução tem sido usado para caracterizar o crescimento dinâmico do software. Durante quase duas décadas foram realizados trabalhos na área,

mas só no final da década de 70 princípio da década de 80, é que começou verdadeiramente o início da história da Evolução de Software. Em 1980 Lehman, após uma série de numerosos estudos a vários sistemas de grandes dimensões, verificou a existência de vários tipos de programas e com isso descreveu a especificação SPE [43]. Esses programas são actualmente considerados também como sistemas. Lehman definiu as chamadas *Leis da Evolução de Software*. O conjunto inicialmente proposto tinha cinco leis [1-5] às quais se juntaram posteriormente [46] mais três[5-8]. Constata ainda que o software acaba sempre por atingir uma certa idade que é mantido ou descontinuado conforme o custo de manutenção *versus* o custo de um novo software.

Apesar da história da evolução de software contar já com 30 anos, existem ainda muitos desafios por resolver [60]. É um subdomínio da Engenharia de Software que investiga possíveis maneiras de adaptar o software para satisfazer especificações de requisitos sujeitas a frequentes actualizações. Contudo, a evolução de software também estuda a mudança do processo em si, analisando as versões que são guardadas nos repositórios para extrair tendências [79], fazer previsões [75] ou compreender a essência do fenómeno da evolução de software [71].

Existem duas vistas da evolução de software que os investigadores frequentemente procuram. São consideradas por Lehman e Ramil em [51] como a perspectiva *"o quê?"* e *"porquê?"* e a perspectiva *"como?"*.

A perspectiva *"o quê?"* e *"porquê?"* foca a evolução de software como uma disciplina científica, que estuda a natureza do fenómeno da evolução de software e procura perceber o seu factor condutor, o seu impacto e outras características relevantes. Esta vista é principalmente tomada em [55].

A perspectiva *"como?"* foca a evolução de software como uma disciplina de Engenharia. Estuda os aspectos mais pragmáticos que ajuda o criador de software ou gestor de projectos nas suas tarefas do dia-a-dia. Esta vista está direccionada para a tecnologia, métodos, ferramentas e actividades que permite os meios para dirigir, implementar e controlar a evolução de software.

De referir também que Lehman et *al.* [51], fazem a distinção, entre aplicar o substantivo evolução e aplicar o verbo evoluir. O substantivo diz respeito à natureza da evolução, as suas causas, propriedades, características, ajudando na resposta a perspectiva *"o quê?"* e *"porquê?"* [18, 38]. O verbo é usado acerca de melhoramento e oferta de meios, processos, actividades, ferramentas, para poder identificar onde a evolução foi implementada e é apontado na procura de resposta a vista *"como?"* [66].

A evolução de software pode ser estudada empiricamente apesar das dificuldades sentidas na aplicação dos métodos e técnicas provenientes de outros domínios quando aplicada a engenharia de software [38].

2.2.1 Especificação Specification Problem Evolution

Lehman propôs uma taxonomia de sistemas [43] que em seguida apresentamos:

A taxonomia identifica 3 tipos possíveis de sistemas de software S, P e E. Um sistema do tipo-S pode ser definido como uma aceitação quando a conclusão do desenvolvimento depende da sua satisfação, no sentido matemático, uma especificação formal. Chamam-se por isso do tipo-S, S de *specification*.

Um programa ou sistema de software do tipo-E, E de *evolution* ajuda a resolver um problema ou realiza actividades do mundo real. Estes tipos de sistemas são propícios a mudança e actualização, logo, a uma evolução.

Por último, a taxonomia identifica os programas do tipo-P, P de *problem*. Esta categoria foi inicialmente introduzida para contemplar a possibilidade de existência de um programa entre os dois tipos anteriores. Mais tarde essa categoria foi considerada redundante uma vez que os programas ou softwares do tipo-P tanto podiam ser do tipo-S como do tipo-E [44, 48, 49].

Posteriormente Cook et *al.* redefiniram essa taxonomia em [22] dando origem à taxonomia *SPE+*. A nova especificação tem como principais inovações, a substituição da definição dos sistemas do tipo-P, passando o "P" a resultar de paradigmas, que deriva do conceito de ciência normal de Khun. A abordagem de Kuhn [40] para a pergunta, "como se desenvolve o conhecimento científico?", é útil para compreender a evolução dos sistemas de software. A sua principal preocupação é explicar um padrão de desenvolvimento do conhecimento que parece característico das ciências como a química e as várias disciplinas dentro da lógica e da matemática. Esse padrão é composto por períodos sucessivos de que Kuhn chama de ciência normal onde cada uma ocorre dentro de uma particular *framework* ou paradigma. Um paradigma neste contexto é definido como, as leis e técnicas que possibilitam a aplicação de suposições teóricas gerais que são adoptadas pelos membros de uma certa comunidade científica [19].

Cook et *al.* definem também que os sistemas do tipo-P e do tipo-S são casos especiais que surgem de tipos de requisitos dos accionistas e reforça explicitamente que o tipo base para os sistemas de software passíveis a evolução são os do tipo-E [22].

A aplicação da especificação SPE e posteriormente SPE+ e do esquema de classificação do software é actualmente reconhecida tornando-se assim uma base fiável da qual futuros trabalhos de investigação poderão apoiar-se.

2.2.2 Princípio da Incerteza do Software

Em [45] Lehman definiu um princípio inerente aos sistemas do tipo-E, chamado de *Princípio da Incerteza do Software* devido ao ritmo de mudança exercido pela dinâmica do mundo real. Na definição desse princípio é explicado que em alguns dos sistemas do tipo-E estão associadas provavelmente suposições inválidas, isto porque, como o mundo real está constantemente a mudar, as suposições tomadas no estágio inicial do desenvolvimento do software estão directamente relacionadas com um estado anterior do mundo real que difere do actual, podendo levar essas suposições a deixar de ter sentido. Algumas suposições não identificadas no início do desenvolvimento do software leva a que nenhuma medida de resolução acerca delas seja tomada nessa altura. Estas suposições implícitas provocarão consequências desconhecidas na execução do programa. O resultado obtido da execução de um programa ou sistema é, assim, incerto. Este princípio continua a ser usado no âmbito da evolução de software (*e.g.* [44, 50, 22]).

2.2.3 Leis da Evolução de Software de Lehman

Na tabela 2.1 são enunciadas as leis da evolução de software propostas por Lehman [43] e completadas em [48] para sistemas do tipo-E referidos anteriormente.

Estas leis são reconhecidas como uma contribuição fulcral no campo da Engenharia de Software. Foram originalmente estipuladas para responder à possível questão de como é que o software evolui durante a sua vida, sujeitas a numerosos estudos [36, 35, 38, 48]. Passados mais de 30 anos ainda são uma fonte de inspiração para a realização de trabalhos na área. O desenvolvimento de leis e teorias da evolução de software é uma tarefa que impõem muitos desafios nos métodos de pesquisa, na exemplificação dos sistemas em estudo, na construção de teorias e nos testes progressivos que permitem a refutação ou refinação de teorias.

Barry et *al.* em [8] classificam as leis em três grandes grupos. As leis 1, 2, 6 e 7 estão relacionadas directamente com as características da evolução de software. As leis 4 e 5 estão ligadas a restrições organizacionais e económicas. As leis 3 e 8 são vistas como "meta-leis", ou seja, leis sobre leis.

Tabela 2.1 Leis da Evolução de Software de Lehman

Nº | Ano	Nome	Lei
I (1974)	Mudança Contínua	Um sistema do tipo-E tem de ser continuadamente adaptado, caso contrário, tornar-se-á progressivamente menos satisfatório de usar.
II (1974)	Aumento da Complexidade	Caso um sistema do tipo-E evolua, a sua complexidade tende a aumentar, a não ser que exista trabalho com o intuito de manter ou reduzir a complexidade actual.
III (1974)	Auto-regulação	Todos os processos evolutivos do sistemas do tipo-E são auto-regulados.
IV (1978)	Conservação da estabilidade organizacional	O ritmo da actividade média nos processos do tipo-E tende a manter-se constante durante a vida operacional do sistema ou fases dessa vida.
V (1978)	Conservação da familiaridade	Em geral, a média do crescimento incremental dos sistemas do tipo-E tende a diminuir.
VI (1991)	Crescimento Contínuo	A capacidade funcional dos sistemas do tipo-E tem de ser aumentada continuamente, para manter a satisfação do utilizador durante a vida útil do sistema.
VII (1996)	Declínio da Qualidade	Embora sejam tomadas rigorosas medidas que visam a adaptação perante a mudança, a qualidade dos sistemas de tipo-E tende a diminuir conforme vai evoluindo.
VIII (1996)	Sistema de Resposta (reconhecida em 1971, formulada em 1996)	Processos evolutivos do tipo-E são sistemas de respostas multi-nível, multi-ciclo e multi-agente.

2.3 Manutenção de Software

A história da *manutenção de software* e do trabalho desenvolvido na área remonta aos anos 60. Desde então, o termo tem sido usado quando são efectuadas modificações deliberadas de partes do software. Só na década seguinte, mais precisamente em 1972, Canning definiu uma classificação num artigo [17] bastante conhecido chamado "*That Maintenance Iceberg*", onde a manutenção é caracterizada apenas como uma operação para a correcção de erros e a ideia de expandir e estender as funcionalidades do software era vagamente pensada.

Em 1976 Swanson [74] propôs, uma tipologia para as actividades de manutenção de software baseada no objectivo dominante ou intenção por parte do dono/utilizador do software. Muitos investigadores adoptaram os termos, "correctiva", "adaptativa" e "perfectiva", mas poucos usam a tipologia tal como o próprio Swanson a define. Em vez disso, cada investigador tem dado significados diferentes a cada termo, não existindo um consenso para os significados dos

termos e suas definições. Esse consenso é importante porque muito do trabalho realizado pelos investigadores é baseado em trabalhos e resultados de outros. Na ausência desse consenso cada investigador define um conceito à sua maneira. A utilização inconsistente de termos pode induzir em erro os investigadores que tentam integrar, comparar ou actualizar resultados obtidos em estudos diferentes. Por exemplo, Polo et *al.* basearam-se no standard ISO/IEC 12207 [2] que deriva do standard ISO/IEC 14764 [3] mas, estes dois standards possuíam definições diferentes para o mesmo conceito [67].

Na tentativa de colmatar essa falta de consenso nos termos e significados, a IEEE estipulou em [1] um glossário que foi revisto em 2006 [5], onde a manutenção de software é definida como modificações de um produto de software após a entrega para corrigir falhas, melhorar a performance ou outros atributos, ou para adaptar o produto a um ambiente modificado.

Esse standard propõem quatro categorias para a manutenção:

- Manutenção Correctiva - consiste na modificação reactiva a um produto de software efectuada após entrega para corrigir falhas que foram descobertas.
- Manutenção Adaptativa - consiste na modificação a um produto de software realizada após a entrega que visa manter a usabilidade de um programa num ambiente modificado ou em mudança.
- Manutenção Perfectiva - qualquer modificação após a entrega do produto que visa melhorar a performance.
- Manutenção Preventiva - refere-se a modificações de software realizadas com o propósito de prever problemas antes que estes aconteçam.

Estas categorias basearam-se em anteriores trabalhos de Swanson e Lientz, onde encontraram a confirmação que a manutenção de software *"consiste em grande parte no desenvolvimento contínuo"* [52]. Desenvolvimento contínuo que muda as funcionalidades ou propriedades do sistema presenciadas pelo utilizador do sistema, que também é chamado de evolução de software [4].

Na história da manutenção de software é importante mencionar o trabalho de Chapin et *al.* no qual os autores estenderam a classificação elaborada por Swanson, mas agora, baseada nas actividades claramente objectivas dos responsáveis pela manutenção que são determinadas através da observação. Podem também incluir problemas não técnicos, tais como documentação,

consulta, instrução, entre outros. Esta extensão apresentada na tabela 2.2 fornece uma maior granularidade do que a classificação anterior, onde doze tipos específicos de manutenção são agrupados em quatro tipos gerais denominados de *clusters* [20].

Tabela 2.2 Clusters e Tipos de Manutenção propostos por Chapin et *al.* [20].

Cluster	**Tipo Específico**
Regras de Negócio	Correctiva, Reductiva e de Melhoramento (do inglês *Corrective*, *Reductive and Enhancive*)
Propriedades do Software	Adaptativa, Performance, Preventiva e *Groomative* (de preparar algo para) (do inglês *Adaptive*, *Performance*, *Preventive and Groomative*)
Documentação	Actualizadora, Reformativa (do inglês *Updative*, *Reformative*)
Suporte Interface	Avaliativa, Consultiva e de Treino (do inglês *Evaluative*, *Consultive and Training*)

Um artigo relacionado com o anterior é o trabalho de Buckley et *al.* [15] onde é proposta uma taxonomia da mudança do software baseado em variadíssimas dimensões caracterizando os mecanismos de mudança e os factores que influenciam esses mecanismos. Essas dimensões são divididas e agrupadas em quatro temas lógicos, propriedades temporais (*quando*), objecto de mudança (*onde*), propriedades do sistema (*o quê*) e suporte a mudança (*como*) e estão representadas na figura 2.1.

A pergunta *quando* visa as propriedades temporais, tais como quando é que uma mudança tem de ser efectuada, que mecanismos são necessários para a suportar. O tema objecto de mudança, tem como contexto as perguntas *onde* do tipo, onde é que no software podemos efectuar mudanças e os respectivos mecanismos necessários para a sua execução. As dimensões do tema propriedades de mudança que visam responder as perguntas *o quê*, agrupam um conjunto lógico de factores que influenciam as mudanças permitidas, como também os mecanismos necessários para suportar tais mudanças que estão directamente ligados as propriedades do sistema de software que está sendo modificado, assim como a plataforma subjacente e o mediador (do inglês *middleware*) em uso. No tema suporte a mudança onde a pergunta *como* é visada, são descritas algumas dimensões, que influenciam os mecanismos necessários para a realização da mudança, ou que podem ser usadas para os classificar.

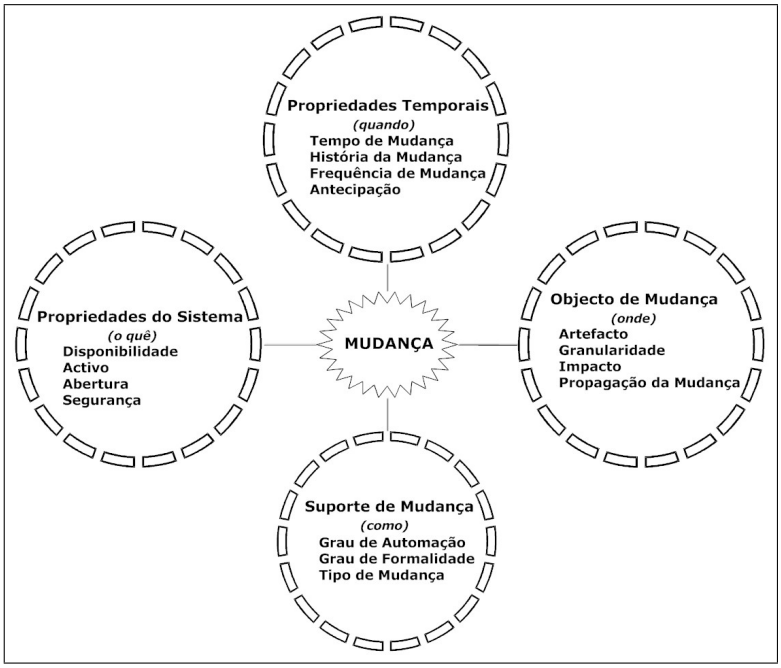


Figura 2.1 Temas e dimensões de mudança do software. Adaptado de [15, 57].

2.4 Associação entre Evolução de Software e Manutenção de Software

A manutenção de software é por natureza uma actividade de desenho. Tende a ser mais exigente intelectualmente e também mais arriscada do que a manutenção de sistemas físicos. Consequentemente, muitos começaram a usar o termo evolução de software como uma alternativa para descrever os vários fenómenos associados com a modificação de sistemas de software existentes.

Este termo tem várias vantagens: primeiro, a evolução incorpora a ideia de mudança essencial que a manutenção simplesmente não conota. Manutenção sugere preservação e correcção, enquanto a evolução sugere novos desenhos evoluindo de velhos. Em segundo lugar, a manutenção é geralmente considerado como um conjunto de actividades planeadas, realiza a manutenção de um sistema. Evolução, considera tudo o que acontece a um sistema ao longo do tempo, para além das actividades previstas, ou fenómenos não planeados que por vezes se manifestam também. Podemos finalmente afirmar que a manutenção e evolução oferecem diferentes perspectivas sobre a natureza da mudança e que existe uma associação entre as duas [20, 35, 38, 58, 69]. Enquanto que a investigação de manutenção de software é mais

direccionada para objectivos da engenharia mais práticos (O que deve ser feito de seguida?), a evolução de software é direccionada para objectivos mais amplos de natureza científica (Como pode um sistema evoluir mais rápido antes que se torne resistente à mudança?).

Muitos autores já devem ter usado o termo "evolução do software", mas é a Lehman e seus colaboradores, que é geralmente creditado como os primeiros a considerar o estudo da evolução de software como fundamentalmente diferente do de manutenção de software [47].

Rajlich e Bennett apontaram que alguns investigadores e alguns profissionais usam o termo evolução de software como um substituto preferido do termo manutenção de software [69]. Tal é também referido por Godfrey e German em [35], mas os autores explicam que a manutenção deve-se a mudanças efectuadas para a resolução de bugs, adaptações do sistema a novos ambientes ou melhoramento do desenho interno do sistema, isto tudo com o intuito de preservar o sistema, enquanto que a evolução resulta da inovação, que favorece e direcciona a mudança para a elaboração de um novo sistema, mais bem adaptado a novos ambientes e que evoluiu do sistema anterior.

Chapin et *al.* definiram a evolução de software como o conjunto de actividades ou processos de manutenção que originam uma nova versão de um software operacional assegurando a qualidade associada às actividades e aos processos, por vezes usada como um sinónimo de manutenção de software. A manutenção de software é definida como a aplicação deliberada de actividades ou processos a um software já existente que visam como principal objectivo a satisfação das entidades interessadas. Referem que a evolução de software ocorre em dois *clusters* da sua classificação, nas regras de negócio e propriedades dos software. Na primeira é quando a manutenção é do tipo correctiva, redutiva ou de melhoramento (do inglês *enhancive*), enquanto que no segundo *cluster* a evolução ocorre apenas quando a manutenção é do tipo adaptativa ou de performance [20].

Kemerer e Slaughter referem que vários investigadores utilizam o termo evolução de software com significados diferentes. Os autores explicam a evolução de software como sendo um processo de análise comportamental do software e como este processo muda ao longo do tempo. A manutenção de software é definida como sendo uma actividade para correcção de erros, modificações necessárias, que permite ao software existente a realização de novas tarefas. Estas actividades ocorrem em qualquer altura após o projecto estar implementado [38].

Podemos finalmente resumir toda a informação dada pelos exemplos, e verificamos que não existe nenhuma definição unânime de evolução de software, nem nos standards da IEEE e nem

nas definições dos investigadores. A evolução de software através dos exemplos dados é um processo que analisa o comportamento do software ao longo do tempo. A presença da palavra inovação é uma constante que origina novas versões de um software que evoluiu da versão anterior. A evolução ocorre quando a manutenção é do tipo correctiva, redutiva, de melhoramento, adaptativa ou de performance considerando o refinamento dado por Chapin et *al.* [20] ou então apenas ocorre quando a manutenção é do tipo correctiva, adaptativa ou perfectiva conforme os standards que a IEEE estipulou [1].

A tabela 2.3 contrasta as características de evolução de software com as de manutenção de software.

Tabela 2.3 Características da Evolução e Manutenção de Software.

		Tipos de Manutenção					
	Origina Novas Versões	Correctiva	Adaptativa	Perfectiva	Preventiva	Inovação	Preservação
Evolução	x	x	x	x		x	
Manutenção		x	x	x	x		x

Neste capítulo discutimos a história da evolução de software e da manutenção de software. Verificamos também que a evolução e a manutenção estão associados, e demonstramos na tabela 2.3 as suas principais características. Revisitaremos este tema no capítulo 4, demonstrando modelos e ferramentas que permitem analisar a evolução de software.

3

Séries Temporais

Este capítulo apresenta uma breve introdução às séries temporais, o que são e para que servem, dando alguns exemplos noutras áreas e apresentando também vários tipos e modelos de séries temporais. São também apresentados trabalhos onde a aplicação das séries temporais é feita no contexto da evolução de software.

Uma série temporal consiste numa sequência de observações recolhidas ao longo do tempo. Muitos conjuntos de dados surgem-nos como séries temporais, tais como, uma sequência mensal da quantidade de bens produzidos numa fábrica ou uma série semanal de números de acidentes na estrada. As séries temporais são usadas no estudo de várias áreas, tais como, economia, engenharia, ciências sociais e naturais (meteorologia e geofísicas).

Um problema intrínseco às séries temporais, é que, tipicamente as observações adjacentes não são independentes. A natureza dessa dependência, através da observação das séries temporais tem um interesse prático considerável. A análise de séries temporais fornece-nos técnicas para a análise dessa dependência.

3.1 Quatro categorias de problemas práticos

Box et *al.* propõem o uso de séries temporais e modelos dinâmicos em quatro áreas de aplicação [14]:

1. A previsão de valores futuros de uma série temporal dados valores anteriores e correntes.
2. A determinação da função de transferência (do inglês *transfer function*) de um sistema sujeito a inércia - a determinação de um modelo entrada-saída (do inglês *input-output*) que permite mostrar no resultado de um sistema o efeito de qualquer série de valores de entrada dada.
3. O uso de variáveis de entrada indicadoras nos modelos de funções de transferência para a representação e avaliação dos efeitos de eventos de intervenção invulgares no comportamento da série temporal.
4. O desenho de esquemas de controlo por meio dos quais, os potenciais desvios dos valores de saída do sistema obtidos a partir de um objectivo pretendido poderão, na medida do possível, ser compensado pelo ajustamento dos valores de entrada da série.

3.1.1 Previsão com séries temporais

O uso no tempo t de observações disponíveis de uma série temporal para prever o valor num tempo futuro $t+l$ pode fornecer uma base para, (i) planeamento económico e de negócio, (ii) planeamento de produção, (iii) inventário e controlo de produção, e (iv) controlo e optimização de processos industriais. As previsões são normalmente utilizadas sobre um prazo (do inglês *lead time*) que varia em cada problema. No contexto das previsões supõem-se que as observações estão disponíveis em intervalos de tempo discretos e equidistantes. Por exemplo, num problema de previsão de vendas de uma loja, as vendas z_t no mês corrente t e as vendas z_{t-1}, z_{t-2}, z_{t-3}, ... nos meses anteriores podem ser usados para a previsão de vendas para os prazos $l = 1, 2, 3, \ldots, 12$ meses seguintes. A previsão feita na origem t das vendas $zt+1$ nalgum tempo futuro $t+l$ é denotada no prazo l por $\hat{z}_t(l)$. A função $\hat{z}_t(l)$, $l = 1, 2, \ldots$, que fornece todas as previsões para a origem t para prazos futuros é denominada de função previsão na origem t. O objectivo é obter uma função previsão tal que o desvio da média do quadrado entre os actuais e os previstos, valores z_{t-l} - $\hat{z}_t(l)$, seja o mais pequeno possível para cada prazo l. Adicionalmente, para calcular as melhores previsões, também é necessário especificar a exactidão

de cada uma, de modo a que por exemplo, os riscos associados às decisões com base nas previsões possam ser calculados. A exactidão das previsões pode ser expressa através do cálculo dos limites de probabilidade nos lados de cada previsão. Estes limites podem ser calculados através de qualquer conjunto de probabilidades. Estes limites são calculados de forma a que os reais valores da série temporal venham a ocorrer dentro dos limites estabelecidos, com a probabilidade determinada.

3.1.2 Estimação das funções transferência

Um tópico de bastante interesse para a indústria é o estudo dos processos dinâmicos, com a finalidade de alcançar um melhor controlo dos processos existentes e melhorar o desenho de novas fábricas. Muitos métodos foram propostos para a estimação da função transferência das unidades fabris desde os registos de processos que consistiam numa série temporal como entrada X_t e uma série temporal de saída Y_t. Os métodos para a estimação de modelos de função transferência baseados em perturbações determinísticas dos dados de entrada, tais como, *step*, *pulse* e mudanças sinusoidais, nem sempre foram bem sucedidos. Isto acontece, quando a resposta do sistema pode estar encoberta por distúrbios incontroláveis referidos colectivamente como ruído (do inglês *noise*).

Outra aplicação importante dos modelos de funções transferência é na previsão. Se por exemplo, for possível verificar-se entre duas séries temporais Y_t e X_t que existe uma relação dinâmica entre ambas, e que, esta relação pode ser determinada, os valores anteriores de ambas as séries podem ser usados para a previsão. Nalgumas situações esta aproximação pode levar a uma considerável redução de erros na previsão.

3.1.3 Análise dos efeitos dos eventos invulgares de intervenção

Nalgumas situações é possível determinar se ocorreu algum evento externo excepcional que possa ter afectado a série temporal z_t em estudo. Por exemplo a incorporação de novos regulamentos ambientais, mudanças em políticas económicas, ataques ou campanhas especiais de promoção. Sobre tais circunstâncias é possível utilizar modelos de funções transferência, discutidos em 3.1.2 para contabilizar os efeitos produzidos na série temporal z_t pelo evento da intervenção. A série de entrada estará na forma de uma simples variável dicotómica tomando apenas os valores 1 e 0 para indicar presença ou ausência do evento.

Nestes casos a análise de intervenção é realizada para obter uma medida quantitativa do

impacto do evento de intervenção na série temporal de interesse. Alternativamente, a análise pode ser empreendida para ajustar algum valor invulgar na série z_t que pode ter resultado de um evento excepcional. Desta forma será assegurado que os resultados da análise das séries temporais, tais como, o ajuste da estrutura do modelo, a estimativa dos parâmetros do modelo e a previsão de valores futuros, não sejam significativamente distorcidos pela influência de eventos invulgares.

3.1.4 Sistemas de controlo discreto

Os aspectos sequenciais do controlo de qualidade têm sido enfatizados, levando à introdução de gráficos de soma acumulada e gráficos da média móvel geométrica. Tais gráficos são frequentemente aplicados na indústria interessando-se pela produção de partes discretas como um aspecto daquilo que é chamado controlo estatístico do processo (do inglês *statistical process control* (SPC)). São usados para a monitorização contínua de um processo. Mais precisamente, são usados para dar apoio a um mecanismo de amostragem contínuo para assinalar as causas detectáveis das variações. Uma amostragem apropriada dos dados de uma fábrica assegura que as alterações significativas são rapidamente chamadas à atenção daqueles que estão responsáveis pela execução do processo. Sabendo as respostas à pergunta "*quando* é que ocorreu uma mudança deste específico tipo?", criamos condições para responder à pergunta "porque é que ocorreu?". O SPC é frequentemente usado como um facilitador da melhoria contínua do processo.

No processo e indústria química, várias formas de ajuste de *feedback* e *feedforward* tem sido usadas naquilo que é chamado de processo de controle de engenharia (do inglês *engineering process control* (EPC)). A este tipo de processos de controle de engenharia, onde os ajustes realizados são usualmente calculados e aplicados automaticamente, é chamado de processo de controlo automático (do inglês *automatic process control* (APC)). Contudo a maneira como esses ajustes são efectuados é uma questão de conveniência. Este tipo de controlo é necessário quando existe distúrbio ou ruído nos dados de entrada do sistema e é impossível ou impraticável a sua remoção. Quando conseguimos medir as flutuações numa variável de entrada que podem ser observáveis mas não mudadas, pode ser possível realizar mudanças compensatórias apropriadas numa outra variável de controlo. Isto é referido como controlo *feedforward*. Alternativamente, ou em adição, podemos ser capazes de usar o desvio do alvo ou "erro de sinal" das próprias características dos dados de saída para calcular mudanças compensatórias apropriadas

na variável de controlo. Isto é chamado controlo *feedback*. Ao contrário do controlo *feed-forward*, este modo de correcção pode ser utilizado mesmo quando a origem dos distúrbios não é totalmente conhecida ou a magnitude dos distúrbios não é medida.

Destas áreas aplicacionais, aquela que é focada nesta dissertação é a utilização das séries temporais para a previsão de valores futuros através de valores passados e correntes.

3.2 Modelos Matemáticos Estocásticos e Dinâmicos Deterministas

Nesta secção vão ser apresentados modelos de séries temporais. Apesar de alguns não serem usados na análise de software, resolvemos apresentá-los pois podem vir a ser utilizados na elaboração desta dissertação.

Modelo estocástico É um modelo matemático cujas variáveis respondem a uma distribuição específica. Estes modelos não oferecem soluções únicas, mas apresentam uma distribuição de soluções associadas a uma probabilidade, segundo uma determinada distribuição de probabilidades. Ou seja, é o modelo matemático que incorpora elementos probabilísticos e que esses elementos representam probabilidades.

Modelo de filtro linear Este modelo estocástico é baseado na ideia de que uma série temporal na qual valores sucessivos são altamente dependentes pode frequentemente ser vista como originada por uma série de ”choques” independentes a_t. Estes choques são gerados aleatoriamente de uma distribuição fixa, assumida usualmente como Normal e que possui média zero e variância σ_a^2. Tal sequência de valores aleatórios a_t, a_{t-1}, a_{t-2}, ... são chamados de processo ruído branco (do inglês *white noise*).

Modelos auto-regressivos Um modelo estocástico que pode ser bastante útil na representação de práticas ocorrentes em séries é o modelo auto-regressivo. Neste modelo, o valor corrente do processo é expresso como um valor finito/limitado, um agregado linear de valores anteriores do processo e um choque a_t.

Modelos de média móvel O modelo auto-regressivo expressa o desvio $\tilde{z}_t$ do processo como uma soma pesada finita de p desvios anteriores $\tilde{z}_{t-1}, \tilde{z}_{t-2}, \ldots, \tilde{z}_{t-p}$ do processo, mais um choque a_t aleatório. Equivalentemente, o modelo de média móvel, expressa $\tilde{z}_t$ como uma soma pesada infinita de a's. Existe também outro tipo de modelo importante na representação de séries temporais observadas, que é o processo média móvel finita onde $\tilde{z}_t$ é tornado linearmente dependente de um número finito q de anteriores a's.

Mistura de modelos auto-regressivo e média móvel Por vezes, quando é pretendido uma maior flexibilidade na elaboração das actuais séries temporais, é vantajoso incluir no modelo termos provenientes dos modelos auto-regressivo e de média móvel. Na prática é frequentemente verdade que uma representação adequada das actuais séries temporais estacionárias ocorrentes pode ser obtida com modelos auto-regressivos, de média móvel ou mistura, nos quais p e q não são superiores a 2 e muitas vezes inferior a 2.

Modelos não-estacionários Muitas séries temporais que se encontram na indústria ou negócio exibem um comportamento não-estacionário e em particular não variam sobre uma média fixa. Tais séries podem mesmo assim exibir um comportamento homogéneo de um tipo. Embora o nível geral sobre que flutuações estão a ocorrer possam ser diferente em tempos diferentes, o comportamento nas séries quando as diferenças de nível são permitidas, pode ser semelhante.

Modelos para função transferência Um importante tipo de relação dinâmica entre uma entrada contínua (do inglês *input*) e uma saída contínua (do inglês *output*), para os quais muitos exemplos físicos podem ser encontrados (Combinar ar e metano de forma a que a mistura dos gases contenha CO_2: entrada: Taxa dos gases, saída: quantidade de CO_2) é aquele em que os desvios de entrada X e saída Y, de valores médios apropriados, estão relacionados por uma equação diferencial linear. Essa ligação é expressa através de uma função transferência $v(B)=\delta^{-1}(B)\Omega(B)$.

Modelos com ruído sobreposto Verificou-se no modelo anterior, que o problema de estimar um modelo apropriado, ligando uma saída Y_t e uma entrada X_t é o equivalente a estimar uma função transferência $v(B) = \delta^{-1}(B)\Omega(B)$. Contudo, este problema é complicado na prática devido a presença de ruído N_t, que corrompe a verdadeira relação entre a entrada e saída

consoante $Y_t = v(B)X_t + N_t$ onde N_t e X_t são independentes. Considera-se que o ruído N_t pode ser descrito por um modelo estocástico não-estacionário do tipo $N_t = \psi(B)a_t = \varphi^{-1}(B)\theta(B)a_t$. Logo na prática é necessário estimar a função transferência $\psi(B) = \varphi^{-1}(B)\theta(B)$ do filtro linear que descreve o ruído, em adição a função transferência $v(B) = \delta^{-1}(B)\Omega(B)$, que descreve a relação dinâmica entre a entrada (do inglês *input*) e a saída(do inglês *output*).

Modelos para sistemas de controlo discreto Como referido na secção 3.1.4, controlar é uma tentativa de compensar os distúrbios que afectam o sistema. Algumas destes distúrbios são mensuráveis; outros não são mensuráveis e apenas se manifestam como desvios inexplicáveis do objectivo da característica a ser controlada. Nestes modelos é estabelecido um esquema que visa a compensação de distúrbios não mensuráveis. Nesse esquema, encontram-se um filtro linear definido como $\psi(B) = \varphi^{-1}(B)\theta(B)$, um processo a ser controlado do tipo $v(B) = \delta^{-1}(B)\Omega(B)$ e um erro do tipo $\varepsilon_t = Y_t - T$, um ruído $N_t = \psi(B)a_t$ e um controlador. A escolha da equação do controlador tem de ser feita de maneira a que o ε tenha o quadrado da média o mais pequeno possível. Um procedimento para o desenho do controlador é equivalente à previsão do desvio do objectivo que poderia ocorrer se nenhum tipo de controlo fosse aplicado e seguindo-se o cálculo dos ajustes necessários para contrariar esse desvio. Segue-se que os problemas de previsão e controlo estão fortemente ligados. Em particular, se um erro de previsão da média quadrada mínimo é utilizado, o controlador produz um erro de controlo de média quadrada mínimo.

3.3 Análise de Software com Séries Temporais

Nesta secção vamos discutir uma série de trabalhos que envolvem a análise de software usando modelos de séries temporais apresentados na secção 3.2.

Fuentetaja e Bagert introduzem o estudo da evolução de software de sistemas usando técnicas de séries temporais. A aplicação de uma versão modificada da análise de flutuação rectificada sobre o conjunto de dados que representa o crescimento incremental de sistemas de software tem demonstrado a presença de fortes anti-correlações de longo termo nos dados. As anti-correlações provam a existência da dinâmica do sistema, uma validação independente da terceira e oitava leis da evolução de software discutidas na secção 2.2 [33].

Em [73], Siy et *al.* propõem uma nova aproximação para representar a história das versões como uma série temporal de *item-set*. Uma série temporal de *item-set* é muito parecida com

a sequência dos dados de uma análise *market-basket* [1]. A única diferença é que a sequência de *item-set* numa série temporal de *item-set* são medidos e gravados em intervalos regulares. Cada medição de um *item-set*, é um *item-set* discreto e por isso, esta técnica é adequada na captura de aspectos não-numéricos de uma história de versões para uma série temporal. Um método baseado em programação dinâmica é descrito para a construção óptima de segmentos de um *item-set* de uma série temporal. A segmentação de uma série temporal representa a série temporal de uma maneira compacta. Ao particionar o período temporal associado as séries temporais em segmentos. De seguida os autores realizam um estudo, em que o foco é o Mozilla. Nos resultados é demonstrado que a segmentação óptima é melhor que a segmentação fixa, nos termos dos números de pessoas que participam no desenvolvimento capturado, as suas actividades e a qualidade das suas modificações dentro do segmento.

Biyani e Santhanam demonstraram que os módulos com mais defeitos no desenvolvimento são provavelmente aqueles que terão mais defeitos após a entrada em produção do produto. Quando é usada a história para prever o número de defeitos durante o desenvolvimento ou após lançamento, sugerem que é suficiente considerar apenas a versão anterior, e afirmam também que o rácio de defeitos no desenvolvimento e após a produção fornece uma maneira de aceder a qualidade relativa de versões de software sem necessitar da informação detalhada no conteúdo do software após o lançamento ou tamanho do código [13].

Muitos dos estudos discutidos nesta secção, utilizam séries temporais para demonstrar a evolução do software, da sua história, daquilo que se passou. Nos relatos de tentativas de evolução encontrados na nossa pesquisa, os autores referem a necessidade de efectuar mais trabalhos na área sobre este particular domínio pois só através de muitos casos de estudo de software se poderá corroborar ou não resultados que alguns autores obtiveram. No que toca a previsão da evolução do software este tema tem tido recentemente bastante atenção por parte dos investigadores havendo a expectativa da obtenção de resultados importantes nesta área, num futuro próximo.

[1] Análise feita por retalhistas para perceber o padrão de compras dos consumidores.

4

Análise da Evolução de Software

A análise da evolução de software assenta em práticas de exploração de repositórios de software. Estes repositórios podem ser encarados como fontes de informação, a partir das quais podemos analisar a evolução do software. De seguida são apresentados modelos e ferramentas que permitem primeiro modelar a informação de forma a que possamos retirar dela as métricas de interesse e de seguida visualizar a evolução do software.

4.1 Repositórios e Sistemas de Rastreio de Defeitos

Os repositórios de software tais como, os sistemas de versões, os sistemas de rastreio de defeitos e os arquivos das comunicações entre os elementos do projecto são utilizados para gerir o progresso dos projectos de software. A exploração dessa informação é cada vez mais reconhecida por parte dos profissionais de software e investigadores como potencialmente benéfica no suporte à manutenção dos sistemas de software, melhoria do desenho e reutilização do software e suporte à validação empírica de novas teorias e métodos. Actualmente a investigação sobre a evolução de software aposta bastante na descoberta de novas formas de exploração desses repositórios, que possam ajudar o desenvolvimento de software, permitindo previsões sobre o mesmo, e planear vários aspectos evolucionários dos projectos de software.

4.1.1 CVS - Sistema de Versões Concorrentes

CVS (*Concurrent Version System*) é o sistema de versões que tem sido o mais usado pela comunidade *open source* nos últimos anos [58].

Possui uma arquitectura cliente-servidor, onde o servidor guarda a versão corrente de um projecto e a sua história, enquanto que o cliente se liga ao servidor para retirar uma cópia integral de um projecto, trabalhar nessa cópia e depois submeter as suas modificações. Foi elaborado com o propósito de guardar várias versões de um mesmo ficheiro. Usualmente este processo necessitaria de muito espaço em disco, mas o CVS guarda todas as versões num único ficheiro. O CVS usa a compressão delta, também chamada de codificação delta, que é uma técnica eficiente, que permite o armazenamento ou transmissão de dados sob a forma de diferenças entre os dados sequenciais e não ficheiros completos. Os clientes podem comparar versões, requisitar a história completa de mudanças, ou retirar um *snapshot* da história do projecto dada uma certa data ou número de revisão. Podem também utilizar o comando *update* que actualiza a cópia local, eliminando a necessidade de repetir o *download* de todo o projecto. O CVS permite também guardar ramos diferentes de um projecto. Por exemplo, enquanto que um ramo de versões é reservado para a correcção de defeitos, outra pode estar em desenvolvimento com mudanças significativas e novas funcionalidades e possibilita ainda o armazenamento dos arquivos binários, mas estes não são tratados de forma eficiente.

O único ficheiro que contém a informação registada pelo sistema de versões chama-se de *log file*. Está representado na figura 4.1 e tem a seguinte estrutura: o número da versão ou revisão, a data da submissão, o autor da submissão, o estado (se o ficheiro ainda está em desenvolvimento ou se foi removido), o número de linhas adicionadas ou removidas consoante a versão anterior, os ramos que a versão corrente tem como raiz e os comentários efectuados pelo autor durante a submissão.

4.1.2 Bugzilla

Bugzilla é um sistema de rastreio de defeitos muito popular entre a comunidade *open source*. O seu núcleo é composto por uma base de dados onde o utilizador pode personalizar a interface *web* e que permite a todos os utilizadores a possibilidade de reportar e seguir a evolução dos problemas num sistema.

Um relatório de defeito do Bugzilla está representado na figura 4.2 e contém a seguinte

```
% cvs log -r1.5 insert-msg.tcl

RCS file: /usr/local/cvsroot/acs/www/bboard/insert-msg.tcl,v
Working file: insert-msg.tcl
head: 1.7
branch:
locks: strict
access list:
symbolic names:
        start: 1.1.1.1
        arsdigita: 1.1.1
keyword substitution: kv
total revisions: 8;     selected revisions: 2
description:
----------------------------
revision 1.5
date: 1999/05/23 01:08:25;  author: philg;  state: Exp;  lines: +42 -22
building new release with bboard fixes and Glassroom and graphics package
----------------------------
revision 1.1
date: 1999/02/27 06:32:09;  author: jsc;  state: Exp;
branches:  1.1.1;
Initial revision
=============================================================================
```

Figura 4.1 Uma fracção do CVS *log file* de `insert-msg.tcl`.

informação: um id que serve para identificar o defeito, o estado do defeito (novo, atribuído, reaberto, resolvido, verificado, fechado) resolução (resolvido, inválido, não resolvem, ainda não, lembrar, duplicado, *worksforme*), o local no sistema identificado pelo produto e o componente, o sistema operativo e a plataforma na qual o defeito foi detectado, uma breve descrição do problema e a lista dos comentários (longa descrição). Muitas vezes um defeito é referente a várias pessoas, a pessoa que o reporta, a pessoa encarregue de resolver o defeito, os responsáveis da qualidade que asseguram a qualidade do software e uma lista de pessoas interessadas em ser notificadas do progresso do processo de resolução do defeito.

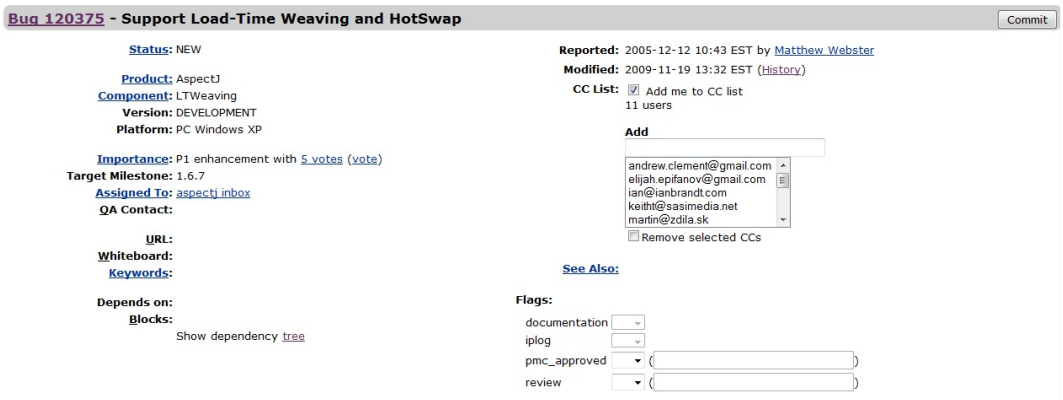


Figura 4.2 Exemplo de um relatório de defeito do AspectJ.

4.2 Modelos e Ferramentas para Análise da Evolução de Software

Esta secção está estruturada da seguinte forma: uma colecção de modelos de dados que são utilizados para a análise de software, seguido de uma tabela resumindo os modelos com as características mais interessantes para o âmbito desta dissertação, e por fim uma estrutura semelhante para as ferramentas de análise utilizadas.

4.2.1 Modelos de dados dos sistemas

Os repositórios guardam grandes quantidades de informação relevante sobre um sistema. Essa informação é vital para a análise da evolução, mas, para efectuar essa análise de uma maneira mais profunda, temos que adicionar informação proveniente dos sistemas de rastreio de defeitos, que contêm a história dos defeitos desse sistema. O problema que tem surgido após a obtenção da informação sobre as versões e sobre os defeitos, é a ligação entre os dois tipos de informação. Os trabalhos e soluções propostas, que vão ser apresentados de seguida, visam a resolução desse problema de ligação entre esses tipos de informação.

Fischer et *al.* em [32], propõem Release History Database (RHDB) que é um modelo de uma base de dados que guarda a informação proveniente dos repositórios e do sistema de rastreio de defeitos, que neste modelo são precisamente, o CVS e o Bugzilla. Este modelo depois de a informação ser importada, processada e inserida, permite facilmente a realização de perguntas simples ou complexas sobre a análise da evolução de software.

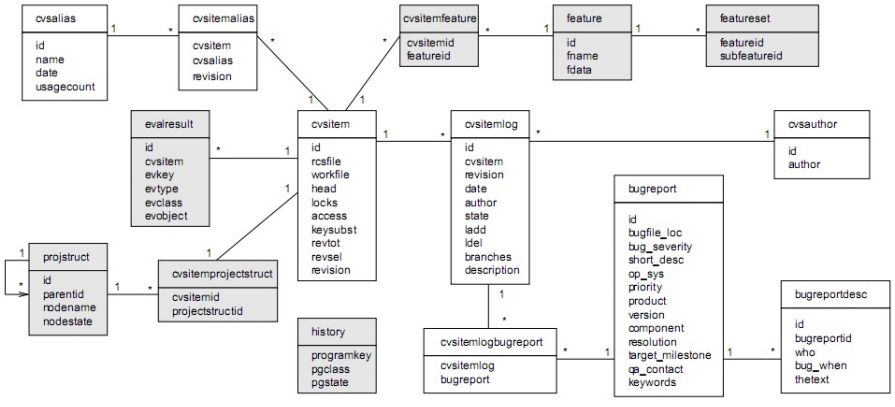


Figura 4.3 Arquitectura de uma RHBD. Adaptado de [32].

Em [76], Čubranić et *al.* propõem o modelo Hipikat. Este modelo, para além de retirar

a informação dos repositórios e sistemas de rastreio, considera a informação existente na documentação guardada nos *sites* dos sistemas analisados como também os *emails* arquivados. A informação guardada pelo Hipikat forma um chamado "grupo implícito de memória" que é usado para a fácil inserção de novos elementos no grupo de desenvolvimento, recomendando artefactos relevantes para tarefas específicas.

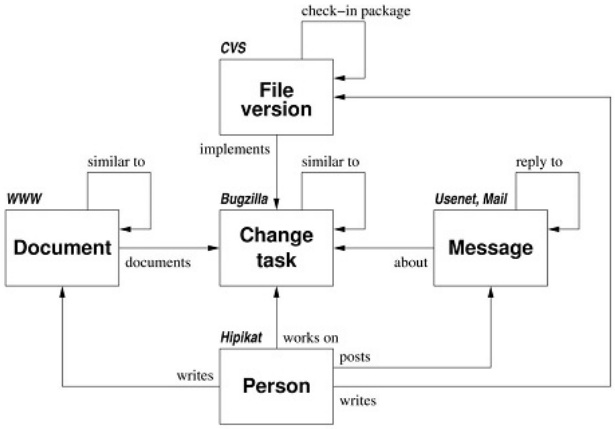


Figura 4.4 Arquitectura do Hipikat. Adaptado de [76].

Outro modelo muito semelhante ao Hipikat, mas que não considera a documentação guardada nos sites dos sistemas é o softChange. Este modelo foi proposto por German et *al.* [34] e, na sua arquitectura destacam-se dois componentes principais, o *Trail Extractor* e o *Fact Enhancer*. O primeiro recolhe a informação, *log files* do CVS, relatórios do Bugzilla, entre outros. O segundo com a informação recolhida gere novos factos que são depois guardados no repositório do softChange e que mais tarde serão utilizados para estatísticas sobre a evolução global do sistema e análise das relações entre ficheiros e autores. Nas estatísticas são usados histogramas enquanto que na análise das relações são usados grafos. Este modelo permite ainda simples visualizações de gráficos, mas concentra-se principalmente na gestão básica e análise dos dados.

Uma aproximação semelhante ao RHDB é o Kenyon, proposto por Bevan et *al.* [12]. O Kenyon fornece uma infraestrutura extensível que permite a extracção de informação de vários repositórios, CVS, SVN, mas não considera a informação dos sistemas de rastreio de defeitos ou dos *emails* arquivados. Este modelo fornece ainda uma interface baseada em *Object-Relational Mapping* (ORM) que permite o acesso aos dados processados e guardados na base de dados.

De salientar que no softChange e Hipikat as tarefas que vão usar os dados já estão definidas enquanto que no Kenyon e RHDB os dados são guardados para futuras análises sobre evolução.

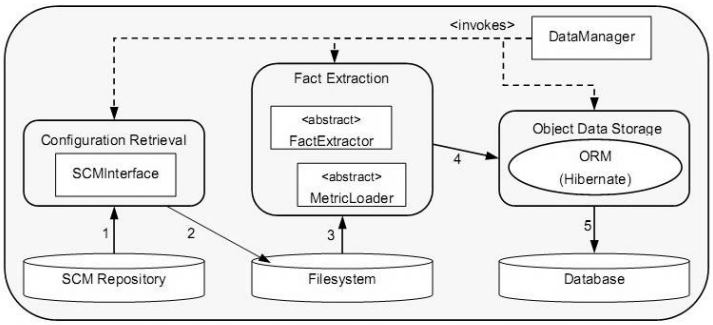


Figura 4.5 Arquitectura do Kenyon. Adaptado de [12].

Na tabela 4.1 encontra-se um breve sumário dos modelos acima descritos bem como algumas das suas características e consequentes diferenças que são importantes na área da evolução de software. De salientar que o Store e Issuezila não são abrangidos pelos modelos, mas serão por ferramentas apresentadas na secção seguinte.

Tabela 4.1 Modelos dos sistemas de software analisados.

		RHDB [32]	Hipikat [76]	softChange [34]	Kenyon [12]
Super Repositórios	SVN				x
	CVS	x	x	x	x
	Store				
Sistemas de Rastreio de Defeitos	Bugzilla	x	x	x	
	Issuezilla				

4.2.2 Ferramentas e técnicas de análise da evolução

Após a extracção da informação proveniente dos repositórios e sistemas de rastreio de defeitos, arquivos de email e documentação disponível e sua inserção num dos modelos acima referidos cujo objectivo é, reduzir o problema de interligação da informação extraída. Ficando em falta a visualização e análise dessa informação. Muitos investigadores ao longo dos anos têm desenvolvido técnicas e ferramentas próprias, de modo a facilitar a visualização da evolução dos sistemas em estudo. Iremos ver ao longo das ferramentas apresentadas que algumas se baseiam em modelos apresentados anteriormente, outras ferramentas dirigem-se a características específicas dos sistemas, enquanto que outras visam a integralidade do sistema.

O primeiro trabalho na área que propõe uma ferramenta para a visualização da evolução de software foi elaborado por Eick et *al.* [30]. O SeeSoft é uma ferramenta de visualização do código fonte, que utiliza cores diferentes para representar fragmentos ou partes do código alterados perante uma modificação. Cada coluna representa um ficheiro enquanto que o código fonte é representado como linhas finas nas colunas, onde a cor de cada linha é determinada através de uma estatística associada ao que o código fonte representa. Ferramentas mais recentes tentam generalizar as diferenças no código fonte entre dois ficheiros, com visões gerais sobre projectos concretos da vida real, que possuam milhares de linhas de código, e que ao longo dos anos várias versões são lançadas.

Collberg et *al.* em [21] desenvolveram o Gevol, é um sistema baseado em grafos para a visualização da evolução de sistemas de software. Cada estado de um sistema é representado por um grafo. São usadas cores para demonstrar as modificações ao longo do tempo tais como, quando partes específicas do sistema foram criadas ou posteriormente modificadas, que programador efectuou modificações e onde, ou que partes do sistema cresceram em complexidade. Todos os nós começam com a cor vermelho, depois tornam-se mais pálidas cada vez que se verifica a ausência de modificações. Quando um nó muda, retoma a cor vermelho. Quando um utilizador repara num evento interessante, tais como, mudanças frequentes num segmento de código, pode clicar num nó para examinar o conjunto de autores que efectuaram essas mudanças. O Gevol considera ainda três tipos de grafos diferentes, herança, controlo de fluxo e grafos de chamadas num programa (*program call graphs* - representam essencialmente as dependências entre as operações que vão sendo chamadas na execução de um programa).

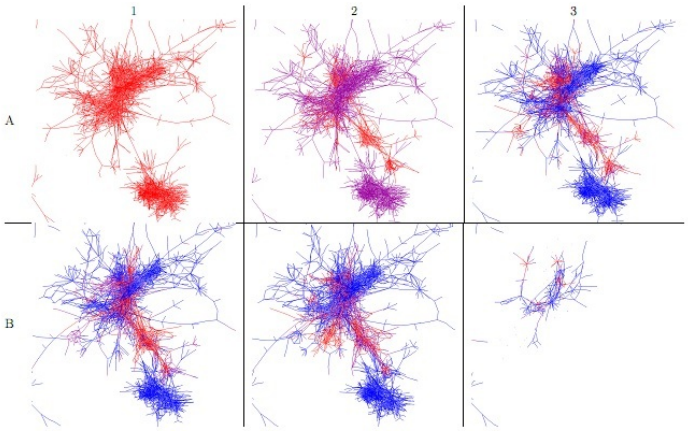
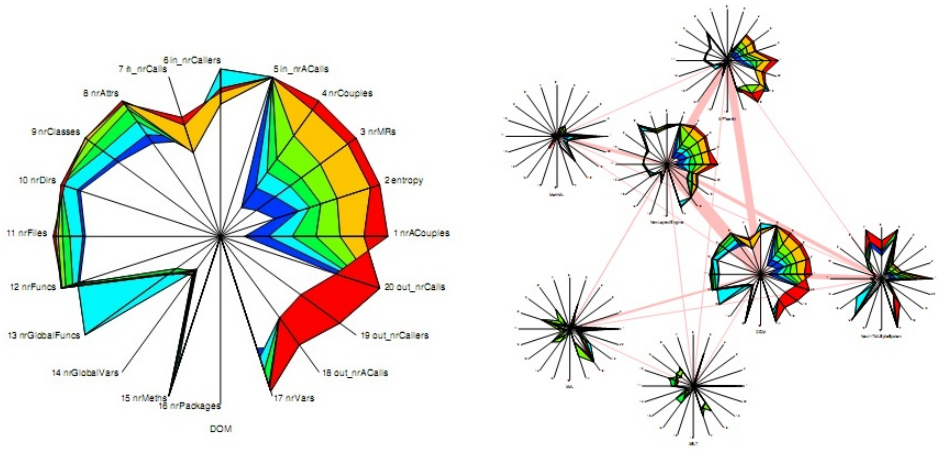


Figura 4.6 Visualização com o Gevol. Adaptado de [21].

Recorrendo também a representação da informação em grafos, Pinzger et *al.* apresentam em [65] uma técnica de visualização chamada de RelVis. Esta técnica baseia-se em diagramas Kiviat (figura 4.7a) e permite a visualização multi-variada de dados tais como o código fonte e métricas de evolução. RelVis quebra o enorme volume de dados extraídos de sistemas de gestão de configuração (CVS) e várias versões do código-fonte de um sistema de software mapeando essa informação em grafos Kiviat (figura 4.7b) que visualizam as medidas através das versões seleccionadas, produzindo dois grafos, um focando as métricas nos nós e outro sobre as métricas de relações de união. Ambos apresentam vistas condensadas do estado corrente da implementação e de como é que a implementação chegou aquele estado.



(a) RelVis: Diagrama de Kiviat. (b) RelVis: Grafo de Diagramas de Kiviat.

Figura 4.7 Visualização com o RelVis. Adaptado de [65]

A Evolution Matrix [41] foi desenvolvida por Lanza e, em vez de utilizar um esboço em árvore, utiliza um esboço com forma de matriz. Esta aproximação mostra a evolução das classes de um sistema, onde cada coluna representa uma versão do sistema, enquanto que cada linha representa versões diferentes de uma classe. Seguindo o princípio das vistas poli-métricas definidas em [42], um número de medidas podem ser mapeadas na largura, altura e cor dos rectângulos que representam as classes. Padrões recorrentes que surgem na matriz levam a uma categorização da evolução das classes. Uma classe que está alternadamente a crescer e diminuir em tamanho é denominada de *Pulsar*. Classes *Pulsar* podem ser vistas como regiões no sistema de intensa actividade onde são feitas mudanças a cada versão do sistema que é lançada. Existem para além deste tipo de classe, as classes do tipo *Supernova* (são classes que subitamente

crescem em tamanho) ou *Dayfly* (são classes que tem um curto período de vida).

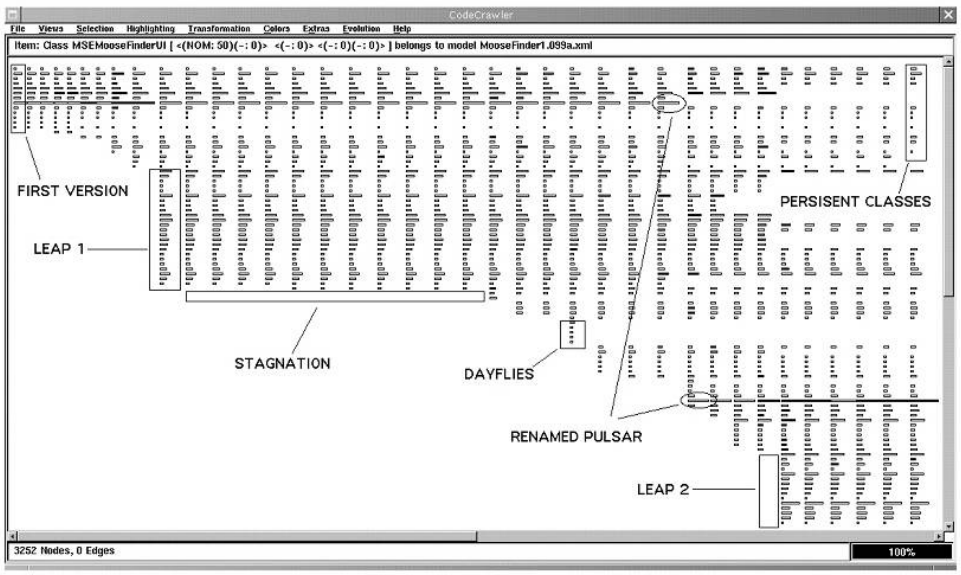


Figura 4.8 Visualização com a Evolution Matrix. Adaptado de [41].

Os investigadores têm tido bastante consideração na análise das dependências de ligação, ou seja, as dependências implícitas entre dois ou mais componentes que normalmente se modificam em conjunto durante a evolução do sistema.

O Evolution Radar proposto por D'Ambros et *al.* em [27], é uma técnica de visualização interactiva para a análise de mudanças que ocorrem conjuntamente que visa a detecção do declínio da arquitectura e identificação de componentes que estão ligados entre si num determinado sistema. Neste caso os componentes podem ser, módulos (grupos) ou grupos de ficheiros (entidades). O módulo em foco é visualizado como um círculo e está colocado no centro de um gráfico em estilo tarte. Os sectores representam os restantes módulos onde o seu tamanho é proporcional ao número de ficheiros contidos naquele módulo. A disposição dos sectores segue a métrica consoante o tamanho, em que o mais pequeno está situado no ângulo 0 (em radianos) e os restantes no sentido dos ponteiros do relógio. Os ficheiros de cada sector estão representados como círculos coloridos e são posicionados segundo as coordenadas polares onde o ângulo e o raio são calculados conforme as seguintes regras:

Radius d (distância desde o centro) É inversamente proporcional a dependência de mudança entre um ficheiro e o módulo em foco. Quanto mais ligados ou dependentes se encontram, mais perto do centro do gráfico está o círculo que representa o ficheiro.

Angle θ Os ficheiros de cada módulo são sorteados alfabeticamente considerando o completo endereço de caminho, e os círculos que os representam são então uniformemente distribuídos nos sectores consoante o ângulo das coordenadas.

Em [28] os mesmos autores utilizam o Evolution Radar para a visualização das dependências de mudança de dois grandes e duradouros sistemas *open source*, onde identificaram alguns problemas como, *God* classes, ficheiros deslocados e dependências em módulos que não são mencionados na documentação. Mostraram ainda uma ligeira integração da técnica num IDE que permite actividades de manutenção como, reestruturação, elaboração de nova documentação, e estimação do impacto das alterações.

Ratzinger et *al.* desenvolveram uma técnica de visualização chamada de EvoLens [70]. Esta técnica é semelhante a Evolution Radar, já que pode ser usada para a análise das relações que geram mudança entre ficheiros de origem e módulos de software. A visualização é feita através da representação por grafos, que permite o utilizador percorrer a informação ao nível dos módulos até aos ficheiros de origem. Toda a informação é retirada do CVS e guardada numa RHDB para sua uniformização e acesso rápido. EvoLens permite ao utilizador a definição de um ponto focal para vistas a lupa e navegar ao longo do tempo através da janela de tempo deslizante definida pelo utilizador. O uso de cor é importante para a compreensão das vistas, pois atribui aos módulos e classes cores diferentes.

Outra ferramenta que utiliza uma técnica de representação por grafos é o EvoGraph apresentado na figura 4.9 e proposto por Fischer e Gall [31]. Esta ferramenta *lightweight* baseia-se nos dados guardados numa RHDB para as análises estruturais e evolucionárias dos sistemas de software. Inspecciona apenas o código fonte que realmente mudou, tornando-se bastante eficiente. Combina os dados das versões com os *log files* do sistema de versões para permitir a avaliação e visualização da estabilidade estrutural e também para suportar a compreensão sobre a interacção entre requisitos evolutivos e desenvolvimento de sistemas.

A técnica do EvoGraph é composta por cinco fases:

Selecção dos ficheiros: Ficheiros de origem que apresentam uma forte dependência de mudança são seleccionados.

Visualização da mudança conjunta.

Extracção de factos: Para os ficheiros seleccionados, a informação detalhada das transacções de mudança são recolhidas da RHDB e são criados vectores de mudança para cada ficheiro envolvido na transacção.

Exploração dos dados: Os vectores de mudança são o ponto de partida para a exploração dos dados das transacções de mudança. Como resultado, é obtida uma descrição da evolução longitudinal das dependências estruturais dos ficheiros escolhidos.

Visualização: As dependências estruturais são visualizadas num diagrama de estilo electrocardiograma [1].

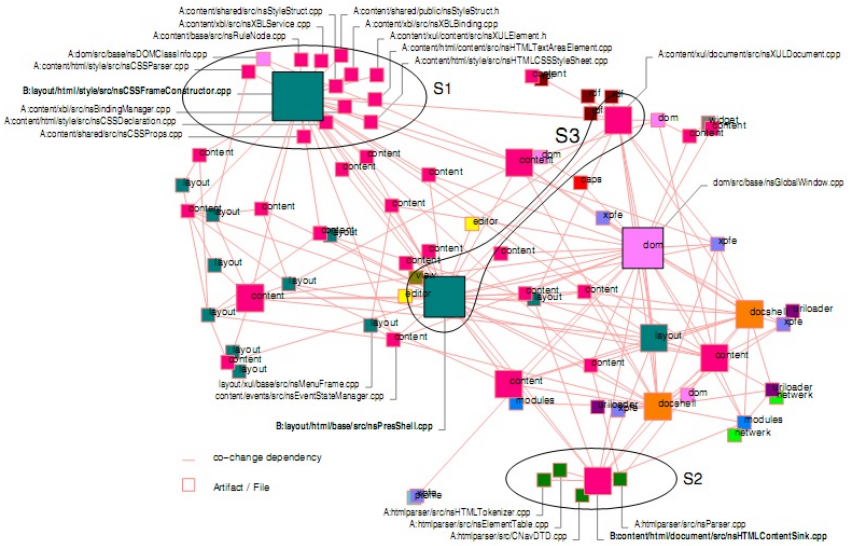


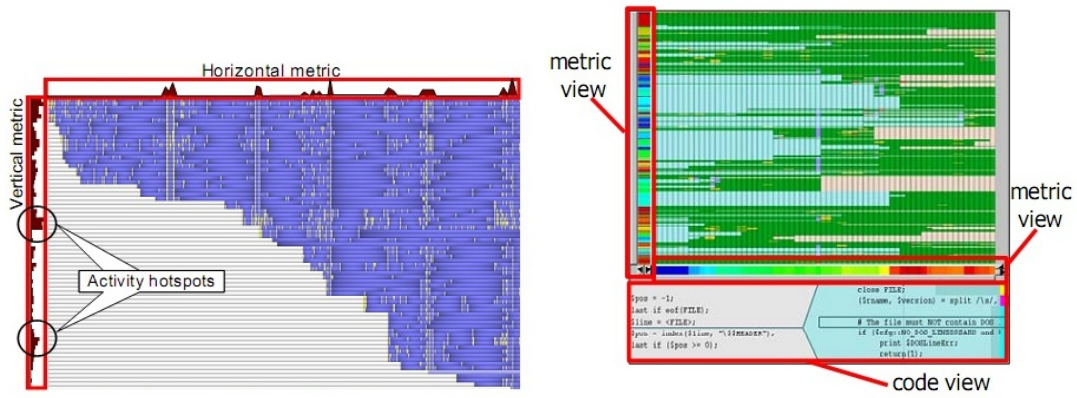
Figura 4.9 Visualização com o EvoGraph. Adaptado de [31].

Voinea e Telea em [77] propõem uma visualização apresentada na figura 4.10a, onde os ficheiros do CVS estão representados, como linhas coloridas onde as cores representam o autor. Esta visualização é implementada no CVSgrab, que é uma ferramenta que permite a visualização e análise de actividades nos repositórios. Aplicaram um algoritmo *cluster* nas visualizações que coloca os ficheiros com desenvolvimento semelhante (consoante cada autor ou actividade) próximos uns dos outros. O objectivo principal deste trabalho foi permitir aos gestores de projectos e programadores que explorassem visualmente a evolução do projecto de uma maneira que a compreensão do sistema e do processo fosse de fácil alcance.

Voinea et *al.* apresentam a ferramenta CVSscan [80] representada na figura 4.10b, em que o processo de extracção dos dados do repositório CVS é baseada no CVSgrab. Esta ferramenta

[1] Electrocardiograma é um exame de saúde na área de cardiologia no qual é feito o registo da variação dos potenciais eléctricos gerados pela actividade eléctrica do coração

permite a visualização da evolução dos ficheiros do CVS através da visualização da evolução de cada linha. O CVSscan permite três tipos de codificação de cores, em que cada cor é associada a uma linha de código e seu autor. É utilizada para compreender quem efectuou modificações e onde essas modificações tiveram lugar, facilitando a compreensão do processo de desenvolvimento.



(a) CVSGrab: Vistas de métricas. (b) CVSScan: Múltiplas vistas do código.

Figura 4.10 Visualização com o CVSGrab e CVSScan. Adaptado de [77] e [80] respectivamente.

Em [24, 26] D'Ambros e Lanza apresentam o BugCrawler que é uma ferramenta de linguagem independente que suporta a evolução de software e engenharia de reversão. É baseada em combinações de métricas de software e visualizações interactivas. Esta ferramenta integra informação estrutural gerada através do código fonte com informação evolucionária retirada dos *log files* do CVS e relatórios de defeitos do Bugzilla. O BugCrawler fornece visualizações *in the large* e *in the small*. As visualizações *in the large* são usadas para obter uma visão geral do sistema em termos de módulos. Módulos que evoluem ao longo do tempo e dependências de módulos. As visualizações *in the small* permitem ao utilizador estudar a estrutura interna individual dos módulos, cobrindo aspectos evolucionários como distribuição do esforço ao longo do tempo entre elementos que estão a desenvolver o software, estabilidade/instabilidade de componentes, entre outras.

Lungu et *al.* propõem o Small Project Observatory (SPO) que é uma plataforma de visualização acessível via Internet. O SPO implementa um catálogo de perspectivas de visualização que são relevantes ao contexto do entendimento dos repositórios ou super-repositórios, é interactivo, apresenta no sistema múltiplas perspectivas ao vivo, fornece detalhes à medida que são

pedidos com um rico conjunto de filtros que endereçam a complexidade [53].

As principais perspectivas oferecidas pelo SPO são:

- Dependência entre projectos - Esta perspectiva apresenta as dependências estáticas entre projectos num super-repositório e permite distinguir os projectos críticos das companhias.
- Colaboração dos programadores - Nesta perspectiva é mostrada como os programadores/*developers* colaboram uns com os outros através das fronteiras de projecto.
- Linhas da actividade do programador - Esta perspectiva apresenta um sumário visual dos períodos em que o programador esteve activo na organização.
- Métricas de evolução - Nesta perspectiva é ilustrada a evolução dos projectos no super-repositório conforme várias métricas.

As perspectivas acima descritas são do interesse dos gestores de projectos, programadores e elementos que asseguram a qualidade. O SPO foi utilizado para a análise de ecossistemas em [54]. Os ecossistemas são definidos como uma colecção de projectos de software que se desenvolvem e evoluem juntos no mesmo ambiente.

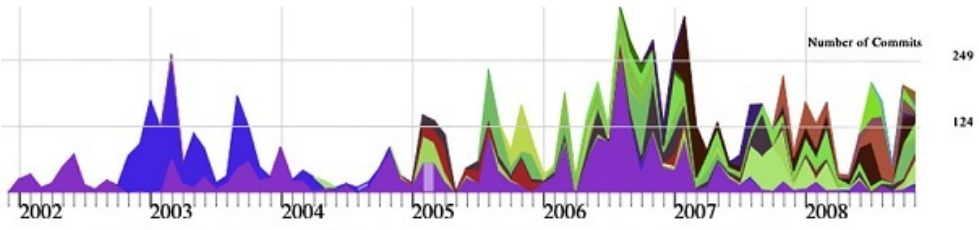


Figura 4.11 Visualização da evolução da actividade em projectos com o SPO. Adaptado de [54].

D'Ambros e Lanza propõem uma ferramenta denominada Churrasco [25], semelhante ao SPO, mas que analisa apenas um sistema, extrai os dados de vários repositórios (CVS e SVN). Por seu lado, o SPO extrai os dados de um único repositório (*Store*) e permite o uso de outras ferramentas, sendo uma delas o Evolution Radar, apresentado na figura 4.12. Esta é uma ferramenta colaborativa que permite a visualização e análise da evolução e as suas principais características são:

Meta-modelo flexível e extensível - O meta-modelo utilizado na ferramenta Churrasco para descrever a evolução do sistema pode ser modificado ou estendido dinamicamente, através do componente meta-base. O meta-base [29] é o componente central de Churrasco

que suporta a flexibilidade e utiliza um componente externo GLORP [2], que é um modulo de mapeamento de objectos relacionais que fornece a persistência na leitura/escrita e de/para a base de dados.

Acessibilidade - A ferramenta Churrasco é totalmente baseada na *web*, permite que a criação do modelo inicial até ao estudo final, e pode ser usada a partir de um navegador *web*, sem ser preciso instalar ou configurar qualquer ferramenta.

Colaboração - Esta ferramenta depende de uma base de dados centralizada e suporta anotações. Assim, o conhecimento do sistema obtido durante a análise, pode ser feito incrementalmente e armazenados no próprio modelo do sistema.

Em [25] os autores demonstram um cenário simples que apoia a Churrasco como uma ferramenta colaborativa de análise da evolução de software.

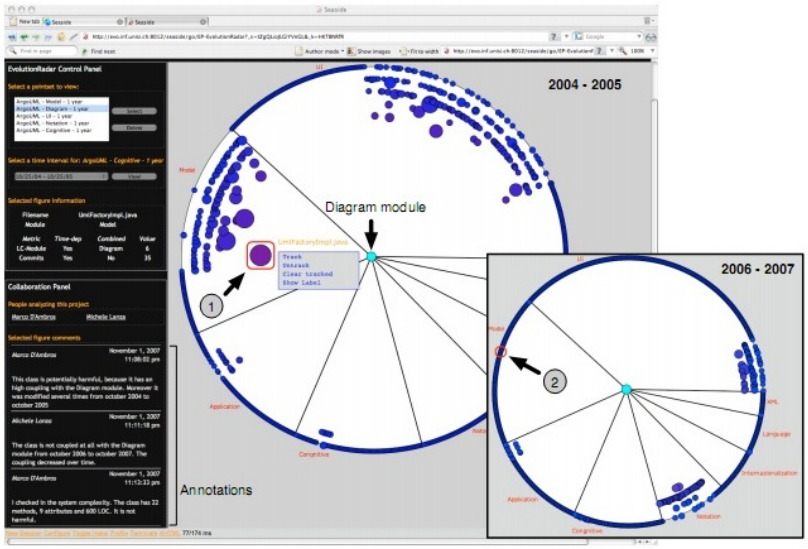


Figura 4.12 Visualização de dois Evolution Radar com o Churrasco. Adaptado de [25].

Na tabela **??** encontramos um breve sumário das ferramentas descritas neste capítulo bem como algumas das suas características e consequentes diferenças que são importantes na área da evolução de software.

[2] Generic Lightweight Object-Relational Persistence

Tabela 4.2 Ferramentas e técnicas de análise da evolução analisadas (Parte 1).

		CVSscan [80]	CVSgrab [77]	SPO [53]	Churrasco [25]	Evol. Radar [27]	BugCrawler [24]
Repositórios	SVN	x	x		x	x	x
	CVS	x	x		x	x	x
	Store			x			
Sist. de Rastreio de Defeitos	Bugzilla				x		x
	Issuezilla						x
Actividades de Análise	Visualização	x	x	x	x	x	x
	Exploração de Dados	x	x	x	x	x	
Granularidade	Sistema		x	x			
	Pacotes				x	x	
	Ficheiros		x	x	x	x	x
	Classes				x	x	
	Linhas	x					
Permite	Snapshot		x	x	x	x	x
	Séries Temporais		x	x	x	x	x
	Previsões		x				
Avaliação da Evolução	Sistema			x			
	Código Fonte		x	x	x	x	
	Dependências					x	

Tabela 4.3 Ferramentas e técnicas de análise da evolução analisadas (Parte 2).

		RelVis [65]	Evol. Matrix [41]	EvoGraph [31]	EvoLens [70]	SeeSoft [30]	Gevol [21]
Repositórios	SVN						
	CVS	x					
	Store						
Sist. de Rastreio de Defeitos	Bugzilla						
	Issuezilla						
Actividades de Análise	Visualização	x	x	x	x	x	x
	Exploração de Dados						
Granularidade	Sistema						
	Pacotes						
	Ficheiros	x	x			x	
	Classes		x				x
	Linhas					x	
Permite	Snapshot	x	x	x	x	x	x
	Séries Temporais		x				x
	Previsões						
Avaliação da Evolução	Sistema						
	Código Fonte	x	x	x			x
	Dependências	x	x	x	x		

4.3 Previsão

Nesta secção discutimos a importância da previsão, quem beneficia da mesma e o que é necessário para se poder efectuar com sucesso. De seguida, apresentamos uma série de trabalhos sobre um tipo de previsão de software, mais propriamente, a previsão de defeitos. Essa previsão, se bem sucedida, é um factor de sucesso, já que permitirá alocar recursos para componentes mais susceptíveis a defeitos, com a finalidade de reduzir os custos de operações de manutenção após a entrega do produto.

Durante a produção de software, o processo que assegura a qualidade consome recursos que são normalmente limitados em tempo e custo. Por isso, é fundamental que o esforço despendido para a realização de operações que asseguram a qualidade seja bem aplicado no sentido de aumentar a eficácia e efectividade desse esforço. Alocar recursos para garantir a qualidade do software é uma tarefa complexa. Por exemplo, direcionar recursos para um módulo sem defeitos durante meses e meses é um desperdício de recursos. Para tal é necessário que o gestor de projecto, coordene e direccione sabiamente, baseando-se na sua experiência, os recursos para onde sejam mais precisos. Antes de se direccionar os recursos, é necessário identificar que partes do software são mais propícias a falhas. No sentido de apoiar os gestores nessa tarefa, foram identificados vários indicadores da qualidade de partes do software.

Uma fonte que pode determinar se um módulo é susceptível a falhas é o seu passado. Se era frequente que uma entidade do software (módulos, ficheiros, entre outros) falhasse no passado, é provável que falhe também no futuro. Tal informação pode ser obtida através dos sistemas de rastreio de defeitos, especialmente quando é ligada a informação obtida dos sistemas de versões ou repositórios, já falados anteriormente no capítulo. Esta informação conjunta permite que seja possível o mapeamento das falhas de entidades específicas. Contudo, previsões precisas requerem uma extensa história de falhas, que pode não existir para a entidade em estudo.

Uma segunda fonte para a previsão de falhas, é o código fonte do próprio programa. Em alguns trabalhos [61, 82, 84] foi demonstrado que métricas de complexidade estão correlacionadas com a densidade de defeitos. Mas o uso indiscriminado de métricas é imprudente, já que é complicado determinar o conjunto de métricas adequado para o projecto em estudo. Outro factor que vai contra as métricas de complexidade é que a maioria das métricas focam elementos singulares, e que raramente tomam em conta a interacção entre elementos.

Nagappan et *al.* efectuam um estudo empírico da história dos defeitos após lançamento (do inglês *post-release*) de cinco sistemas de software da Microsoft [3]. Neste estudo os autores verificaram que, as entidades do software propícias a falhas estão estatisticamente correlacionadas com medidas de complexidade do código. Salientam que não existe um único conjunto de métricas de complexidade que possa actuar como o melhor preditor de defeitos. Por isso, recorrem a análise de componentes principais. Uma técnica estatística que permite aglomerar várias métricas numa métrica "sintética" que em princípio captura o essencial das várias que nela se juntam. Construíram modelos de regressão que prevêem com precisão a probabilidade de defeitos após lançamento para novas entidades [61].

Em [82] Zimmermann e Nagappan, propõem o uso da análise de redes sobre grafos de dependências. Esses grafos representam as dependências que existe entre vários pedaços de códigos, das quais o gestor de projectos pode ter pouco conhecimento. A análise de redes sobre esses grafos, permitirá ao gestor identificar unidades centrais do programa susceptíveis a defeitos. Como exemplo de uso dessa análise, os autores realizaram a avaliação do Windows Server 2003 [4], e encontraram que o *recall* (mede a percentagem de binários observados como propícios a defeitos que foram correctamente identificados) para modelos construídos de medidas de redes é superior do que nos modelos construídos de métricas de complexidade. Verificaram também que as medidas de rede conseguiram identificar 60% dos binários que os *developers* consideraram como críticos, o dobro dos identificados pelas métricas de complexidade.

Schröter et *al.* realizam um estudo empírico sobre 52 *plug-ins* do Eclipse, onde descobrem que o desenho do software, bem como, o histórico de falhas do mesmo, podem ser usados para construir modelos que prevêem correctamente se um componente é propício a falhas num novo programa. Os autores utilizam as relações entre componentes definidas tipicamente na fase de desenho para realizarem as suas previsões, deste modo os desenhadores podem facilmente explorar e ter acesso a alternativas previstas de qualidade. Como resultado deste estudo, foi verificado que 90% dos componentes que estão entre os 5% de componentes mais propícios a falhas foram previstos pelo modelo, contra apenas 33% num palpite aleatório [72]. Foi feita a replicação deste trabalho em [83] por Zimmermann e Nagappan, mas o software alvo para o estudo foi o Windows Server 2003.

Outro trabalho também sobre o Eclipse foi realizado por Zimmermann et *al.* em [84]. Neste trabalho os autores realizaram o mapeamento dos defeitos extraídos do sistema de rastreio de

[3] www.microsoft.com

[4] Windows Server 2003 - sistema operativo para servidores produzido pela Microsoft

defeitos do Eclipse nas partes do código fonte onde ocorreram. As versões do Eclipse analisadas neste trabalho foram a 2.0, 2.1 e 3.0, onde foi gerada uma lista de defeitos antes e depois do lançamento, numa janela de tempo de 6 meses, isto, para cada pacote e ficheiros de cada versão. Os autores enriqueceram ainda mais os dados com métricas comuns de complexidade. Como resultado foi verificado que a combinação de métricas de complexidade pode prever defeitos, sugerindo que quanto mais o código se encontra complexo, mais defeitos tem.

Um estudo um pouco diferente dos anteriores é o de Panjer, onde prevêem o tempo que vai ser necessário para resolver um defeito. Também é utilizado o Eclipse, mais propriamente o Bugzilla do Eclipse já mencionado em 4.1.2 onde estão guardados todos os defeitos do Eclipse reportados. Um conjunto de dados de defeitos do Eclipse foi fornecido por Zimmermann, sendo posteriormente enriquecidos com mais informação sobre o histórico de mudanças. Após a exploração e avaliação dos dados recorrendo a técnicas de exploração de dados, o autor verificou que pode ser alcançada uma precisão de 34.9% recorrendo apenas ao uso de atributos primitivos associados a um defeito. É referido ainda que os factores que influenciam mais a duração da resolução de um defeito são, actividade de comentários, a severidade dos defeitos determinada pela equipa de desenvolvimento do projecto, produto, componente e versão [63].

5

Eclipse

Neste capítulo é apresentado o software do qual será feito o estudo da evolução especificando algumas das suas características principais.

O Eclipse [1] é uma plataforma *open source* extensível desenvolvida em Java [2], para a construção de ambientes integrados de desenvolvimento IDE (*Integrated Development Environment*) sendo ele próprio um IDE.

Fornece um conjunto de serviços que permitem o controlo de ferramentas funcionando em simultâneo, apoiando as tarefas de programação. Os criadores de ferramentas contribuem para a plataforma Eclipse envolvendo-as em componentes conectáveis chamados *plug-ins* que poderão ser descarregados pelo utilizador. O mecanismo de extensibilidade no Eclipse ocorre quando um novo *plug-in* pode adicionar novas funcionalidades a um *plug-in* já existente, fornecendo assim na inicialização deste processo um conjunto fundamental de *plug-ins* que visa permitir o bom funcionamento e integração dos novos *plug-ins* com os *plug-ins* já instalados no Eclipse.

Actualmente é um dos IDE Java mais utilizados, onde o núcleo dos seus conceitos e estrutura permitem um modelo geral para a elaboração de aplicações onde partes constituintes são desenvolvidas por várias pessoas.

[1] http://www.eclipse.org/
[2] http://java.sun.com/

A utilização do Eclipse traz a vantagem da visibilidade e da aceitação industrial e ainda permite a reutilização de componentes disponibilizados por uma vasta comunidade de contribuidores. Contudo existe uma desvantagem da falta de controlo sobre os versões lançadas. Nem todas as versões permitem o bom funcionamento de todos os *plug-ins*, daí a necessidade de criar um guião de transição com as incompatibilidades entre versões [3]. Por essa razão, é necessário às vezes, manter várias versões do Eclipse. Para além da funcionalidade que permite ao Eclipse ser estendido para utilizar outras linguagens de programação como C ou Python, esta estrutura baseada em *plug-in* permite ao Eclipse trabalhar com vários tipos de aplicações, tais como LaTeX, sistemas de gestão de base de dados e aplicações de rede. O suporte de Java e CVS é fornecido no Eclipse SDK (*Software Development Kit*), o suporte para SubVersion é fornecido por *plug-ins* de terceiros.

Bauer e Pizka referem que o desenvolvimento de software *open source* é uma boa fonte de conceitos e princípios da evolução de software [9]. De salientar também a importância da escolha do Eclipse como objecto de estudo, visto que o Eclipse é um projecto real e de grande dimensão, o que confere um maior impacto aos resultados a obter neste estudo.

5.1 *Plug-in*

Um *plug-in* no Eclipse é um componente que fornece um certo tipo de serviço num contexto disponibilizado na interface gráfica do mesmo. Ao executarmos o Eclipse, uma infraestrutura de gestão dos *plug-ins* que é invisível ao ambiente onde as actividade de desenvolvimento são realizadas, é disponibilizada para que todos os *plug-ins* possam funcionar correctamente. Nesse processo é permitida a adição, remoção e actualização dos mesmos.

Um *plug-in* é descrito num ficheiro XML (*eXtensible Markup Language*), onde está contida a informação que é fornecida ao Eclipse durante a sua execução para saber do que necessita o *plug-in* para a sua activação. Esse ficheiro é chamado de manifesto e está representado na figura 5.1.
No Eclipse um *plug-in* pode estar relacionado com outro *plug-in* por uma de duas relações:

Dependência - Os papéis nesta relação são, *plug-in* dependente e *plug-in* pré-requisito. O *plug-in* pré-requisito suporta as funções do *plug-in* dependente.

[3] http://dev.eclipse.org/viewcvs/index.cgi/org.eclipse.platform.doc.isv/porting/3.5/incompatibilities.html

Extensão - Os papéis nesta relação são, *plug-in* anfitrião e *plug-in* extensão. O *plug-in* extensão estende as funções do *plug-in* anfitrião.

Estas relações são especificadas no manifesto do *plug-in* através dos campos XML *requires* e *extension*.

```
<?xmlversion="1.0" encoding="UTF-8"?>
<plugin
id="com.example.hello"
name="Hello Plug-in"
version="1.0.0"
provider-name="EXAMPLE"
class="com.example.hello.HelloPlugin">
<runtime>
   <library name="hello.jar"/>
</runtime>
<requires>
   <import plugin="org.eclipse.core.resources"/>
   <import plugin="org.eclipse.ui"/>
</requires>
<extension
      point="org.eclipse.ui.perspectiveExtensions">
   <perspectiveExtension
         targetID="org.eclipse.ui.resourcePerspective">
      <actionSet
            id="com.example.hello.actionSet">
      </actionSet>
   </perspectiveExtension>
</extension>
</plugin>
```

Figura 5.1 Manifesto de um *plug-in* em XML.

5.2 Arquitectura do Eclipse

A arquitectura baseada em *plug-ins* apresentada na figura 5.2 é um mecanismo leve que suporta o carregamento de qualquer extensão desejada para o ambiente. Permite gerir a sua própria configuração, sendo por isso, possível a inclusão de *plug-ins* durante o tempo de execução, ao contrário de outras aplicações, que carregam todas as funcionalidades que poderão ser usadas no início. Todos os componentes no Eclipse são *plug-ins*, com a excepção do pequeno núcleo do tempo de execução.

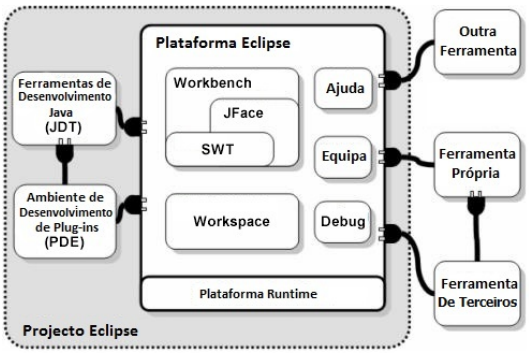


Figura 5.2 Arquitectura do Eclipse. Adaptado do site do Eclipse [1].

Isto significa que cada *plug-in* criado que integra o Eclipse segue um procedimento igual aos outros *plug-ins*, ou seja, todas as funcionalidades são criadas da mesma maneira. O Eclipse fornece *plug-ins* para uma ampla variedade de funcionalidades, algumas das quais através de terceiros recorrendo a utilização de modelos comerciais e livres.

5.3 OSGi service platform release 4

A arquitectura do sistema de tempo de execução do Eclipse é baseada no Equinox [4], que é uma implementação que segue as especificações e standards da OSGi [5] implementadas no *OSGi Service Platform Release 4* representado na figura 5.3.

O OSGi foi inicialmente pensado para suportar aplicações de automatização residenciais. Consiste numa pequena camada acima da máquina virtual do Java, que fornece uma plataforma partilhada para o provisionamento de componentes e serviços de rede. Fornece um modelo extensivo de segurança e ao mesmo tempo promove a cooperação e reutilização entre componentes. As características mais interessantes da plataforma são, como já referido, o desenho do tempo de execução baseado numa arquitectura orientada a serviços e a capacidade de suportar actualizações dinâmicas sem perturbar o ambiente em execução.

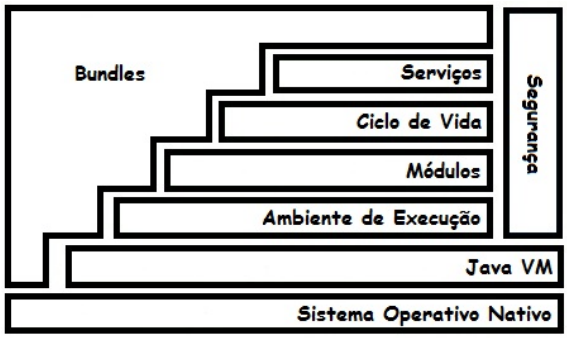


Figura 5.3 Arquitectura do OSGi Service Platform Release 4. Adaptado do site da OSGi [5]

A lista seguinte contém uma breve definição dos termos:

Bundles - *Bundles* são os componentes OSGi disponibilizados pelos seus criadores.

[4] http://eclipse.org/equinox/
[5] http://www.osgi.org/Main/HomePage

Serviços - A camada de serviços que faz a ligação dos *bundles* de uma forma dinâmica oferecendo um modelo divulga-descobre-liga para objectos Java.

Ciclo de Vida - A API (*Application Programming Interface*) que instala, inicia, pára, actualiza e desinstala *bundles*.

Módulos - A camada que define como é que um *bundle* pode importar e exportar código.

Segurança - A camada que trata dos aspectos de segurança.

Ambiente de Execução - Define que métodos e classes estão disponíveis numa plataforma específica.

Neste capítulo foi apresentado o IDE Eclipse. Discutimos a sua arquitectura e os componente em que se baseia essa mesma arquitectura e discutimos também as razões que justifiquem a escolha deste software como nosso objecto de estudo.

6

Análise e Previsão da Evolução do Eclipse com Séries Temporais

Contexto - Um gestor de projectos tem de tomar decisões que influenciarão o desenvolvimento do software no futuro. Neste contexto, e sabendo que qualquer tomada de decisão terá sempre importantes repercussões no futuro do software, têm sido sugeridas por muitos investigadores as mais adequadas práticas, orientações, técnicas e ferramentas. Algumas dessas ferramentas permitem a análise e previsão da evolução do software. Se esta capacidade de prever for alcançada, será sem dúvida um auxiliar importante para a gestão do processo de evolução, a ser usada pelos gestores de projectos.
Objectivos - Neste capítulo apresentamos um estudo observacional em que fazemos uma análise e previsão da evolução do Eclipse com séries temporais.
Métodos - Usamos séries temporais do número de defeitos do Eclipse, submetidos ao sistema de rastreio de defeitos Bugzilla, para a construção de modelos de previsão ARIMA.
Resultados - Observamos que a utilização de modelos ARIMA para a previsão de defeitos, é uma aproximação válida, identificando padrões sazonais e tendências nas séries temporais usadas nos modelos ARIMA. As previsões obtidas têm uma margem de erro aceitável o que as torna suficientemente fiáveis.

Limitações - Através da realização desta experiência verificamos que para a utilização dos modelos ARIMA seja a melhor e mais eficiente possível, é necessário uma razoável quantidade de dados históricos do software para que o modelo se possa ajustar da melhor forma a série temporal. Daqui resulta que os modelos aqui propostos não são adequados numa fase mais inicial de um projecto, em que ainda não tenham sido recolhidos dados históricos suficientes.
Conclusões - Concluímos que a utilização dos modelos ARIMA é uma aproximação válida para a previsão do número de defeitos do Eclipse. Os resultados obtidos neste estudo são consistentes com resultados de um estudo anterior que usava séries temporais para avaliar *trouble tickets* no contexto projectos de desenvolvimento de uma grande empresa produtora de software. Esta observação reforça a nossa confiança na validade externa dos resultados obtidos nesta dissertação.

6.1 Motivação

6.1.1 Descrição do Problema e Contexto

Numerosos estudos científicos sobre grandes sistemas de software têm demonstrado que grande parte do esforço realizado pelos elementos de desenvolvimento e o custo despendido em grandes projectos de software revertem para a manutenção e evolução dos sistemas de software existentes, ao contrário do desenvolvimento de um projecto de raiz [10, 23, 78]. Tal facto deve-se principalmente à necessidade de evolução progressiva do software de sistemas em função dos requisitos que vão mudando no tempo.

Na tentativa de reduzir o esforço e custos das operações de manutenção e evolução de sistemas existentes, muitos investigadores têm sugerido boas práticas, orientações, técnicas e ferramentas que analisam o software e demonstram os problemas presentes no software. No entanto, é menos frequente os investigadores tentarem prever o comportamento futuro do software e onde será necessária maior intervenção de operações de manutenção e evolução. Esta capacidade de prever, se alcançada, será um auxiliar importante para a gestão do processo de evolução.

6.1.2 Questões de Investigação

O objectivo do trabalho é, essencialmente, dotar o gestor do processo de evolução de uma ferramenta que lhe permita fazer previsões sobre a evolução do software, não só porque lhe permite controlar melhor o processo, mas também porque lhe poderá ajudar a identificar oportunidades de melhoria do mesmo processo. Neste sentido tentamos encontrar factos que respondem às seguintes Questões de Investigação (QI):

QI1: Será que a distribuição do número de defeitos nas séries temporais analisadas exibe algum padrão de sazonalidade?

Esta questão procura responder se as diferentes séries temporais analisadas exibem algum padrão periódico, e se sim, identificar a razão desse padrão sazonal.

QI2: Será que a distribuição dos números de defeitos na série temporal analisada exibe alguma tendência particular?

Esta questão procura responder se a série temporal exibe alguma tendência, e se sim, identificar a razão dessa tendência.

QI3: Será que a previsão baseada nos modelos ARIMA é uma aproximação válida para prever a evolução do número de defeitos?

Esta questão procura responder se a previsão com modelos ARIMA é melhor que um modelo de caminho aleatório, sendo para tal efectuada a comparação entre os dois modelos.

QI4: Qual é a precisão das previsões dos modelos ARIMA para o número de defeitos?

Esta questão procura analisar os erros ocorridos aquando da previsão dos modelos ARIMA. Para isso, são efectuados cálculos que nos indicam a precisão e fiabilidade das previsões.

QI5: Será que existe algum padrão sazonal para a resolução dos defeitos ao longo do tempo?

Esta questão procura responder se existe algum padrão periódico na resolução dos defeitos ao longo do tempo, bem como a eventual existência de alguma relação com os padrões sazonais das séries temporais.

QI6: Será que existe alguma tendência para a resolução dos defeitos ao longo do tempo?

Esta questão procura responder se existe alguma tendência na resolução dos defeitos ao longo do tempo, bem como a eventual existência de alguma relação com a tendência das séries temporais.

QI7: Será que a eficiência dos modelos ARIMA se mantém, numa janela temporal de dimensão fixa de pontos de dados da história do software quando deslocamos o ponto inicial dessa janela?

Esta questão procura responder se um modelo ARIMA continua a ser eficiente, alterando os limites do intervalo dos pontos de dados, mas mantendo a distância entre esses limites, de forma a identificar em que momento é oportuno utilizar este tipo de modelos.

6.1.3 Objectivos do Trabalho

O nosso objectivo é analisar os defeitos do Eclipse, procurando padrões e tendências, com a finalidade de caracterizar a sua evolução ao longo do tempo e permitir a criação de modelos de previsão para que possam ser utilizados pelos gestores de projectos. Este estudo é realizado num contexto de um estudo observacional sobre os defeitos do Eclipse disponíveis através do Bugzilla.

6.2 Trabalho Relacionado

Raja et *al.* em [68] realizaram o primeiro trabalho sobre a análise de séries temporais aos defeitos de um projecto multi-organizacional relatados durante a evolução do software. Os autores utilizaram o modelo de séries temporais *ARIMA(p,d,q)* onde os valores *p*, *d* e *q* são normalmente os inteiros 0 e 1, mas podendo tomar também o valor 2. O modelo é parametrizado como *ARIMA (p,d,q)*, onde *p* é a ordem do componente auto regressivo; *d* é a ordem do componente diferenciado, e *q* é a ordem do componente de média móvel. Estes autores descobriram que, ao contrário das investigações anteriores, em que existia a necessidade de adaptar modelos para um contexto individual de desenvolvimento, um simples modelo *ARIMA(0,1,1)* prevê com precisão os padrões típicos do número de defeitos na evolução de software de oito projectos *open source*, desenvolvidos, mantidos e geridos independentemente. O modelo ARIMA enquadra-se nos modelos não-estacionários apresentados no capítulo 3. Este modelo pode ser usado para avaliar e comparar a fiabilidade das soluções candidatas *open source*, e para facilitar o planeamento do orçamento da evolução do software e distribuição do tempo.

Bernstein et *al.* realizaram uma experiência cujo objectivo foi identificar que características temporais dos dados são centrais no desempenho de modelos de previsões [11]. Sugeriram também que o uso de modelos não lineares, contrariamente aos tradicionais modelos de regressão linear, permitem a descoberta de inter-relações escondidas entre as características temporais e os defeitos. A experiência consistiu no estudo de seis *plug-ins* do Eclipse. Para tal, recolheram a informação desses *plug-ins* no Bugzilla e CVS, calcularam de seguida um conjunto de métricas relacionadas com o código fonte da versão mais recente disponível à data, bem como de defeitos do passado e modificações, de onde foram retiradas as métricas mais relevantes. Aplicaram de seguida um sistema de aprendizagem baseado em árvores de decisão (do inglês *decision tree learner*) para a previsão da localização dos defeitos. Para a previsão do número de defeitos, os autores aplicaram um sistema de aprendizagem baseado em árvores de regressão (do inglês *regression tree learner*) ao invés da tradicional regressão linear. Estes autores concluíram que o uso das características temporais melhora significativamente o desempenho dos modelos de previsão.

Um exemplo do uso de séries temporais para estudar a evolução de um software é realizado por Mens et *al.* [59], no qual os autores escolheram um conjunto de métricas sobre o Eclipse (figura 6.1). Uma das finalidades desse trabalho foi tentar verificar algumas das leis de evolução de software de Lehman, mais precisamente as leis 1, 2 e 6, descritas na tabela 2.1. A informação para análise foi recolhida ao longo de 7 versões e várias versões intermédias lançadas entre 2001 e 2007 e, após o seu processamento e análise, os autores verificaram que a evolução do Eclipse obedece às leis 1 e 6, embora não tenha sido possível verificar de forma conclusiva a adesão à lei 2. Essa não total verificação deve-se em grande parte à dificuldade em definir a complexidade de um sistema e também a limitações eventualmente impostas pelas ferramentas usadas no estudo da evolução. Neste caso a ferramenta STAN [1], não considera métricas de acoplamento e coesão. Estas métricas são normalmente usadas para estudar modularidade do software. Tendo em conta a importância da modularidade de software na facilidade de evolução, sendo mesmo considerada um dos mais relevantes atributos de qualidade neste contexto, há razões fortes para acreditar que a inexistência de suporte a métricas de modularidade na ferramenta STAN terá sido, de facto, uma limitação importante na avaliação da 2ª lei de Lehman, pelo que essa avaliação ficou, assim, em aberto.

[1] http://stan4j.com/

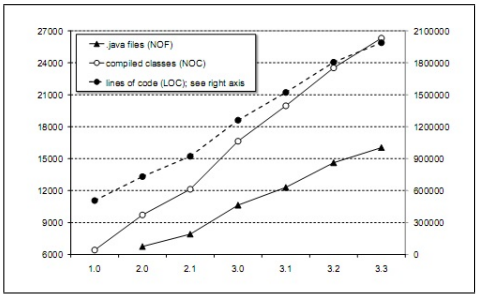


Figura 6.1 Exemplo de série temporal, onde se verifica o crescimento do número de ficheiros(NOF), de classes(NOC) e linhas de código(LOC) de várias versões do Eclipse. Adaptado de [59].

Wermelinger et *al.* usou séries temporais com o propósito de estudar a evolução de software [81]. Nesse trabalho, é também analisado o IDE Eclipse, com o objectivo de investigar como são usados na prática os princípios de desenho estrutural, a fim de avaliar a utilidade e relevância de tais princípios para a manutenção de sistemas grandes, complexos, duradouros e bem sucedidos. As métricas que os autores escolheram para o estudo são baseadas sobre os *plug-ins*, que são fundamentais na arquitectura do Eclipse. Mais precisamente, essas métricas incidem sobre o número das dependências dos mesmos. Essas dependências são caracterizadas como internas, externas, estáticas ou dinâmicas. Um exemplo dos resultados obtidos é representado na figura 6.2, em que podemos observar do lado esquerdo a evolução de métricas ao longo das várias versões principais e do lado direito a evolução ao longo das várias *milestones* entre duas versões principais. Após a análise dos dados, verificaram que a arquitectura do eclipse não segue a norma aconselhada do aumento de coesão. Estas descobertas podem ser úteis nas práticas de desenvolvimento e de manutenção do software particularmente em sistemas que requerem uma arquitectura onde a coesão vai diminuindo mas com um núcleo estável e extensível.

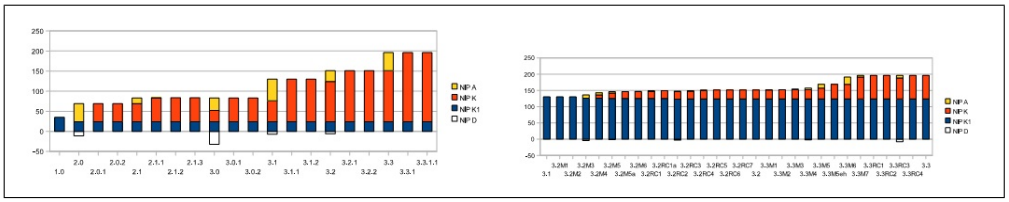


Figura 6.2 Exemplo de série temporal, onde se verifica a evolução de métricas de várias versões do Eclipse. Adaptado de [81].

Kläs et *al.* apontam para a falta de métodos empíricos válidos que visam a construção de modelos de previsão específicos por contexto para o planeamento e controlo das actividades das

equipas responsáveis pela qualidade nas fases iniciais do ciclo de vida de um software, quando os dados históricos e os dados de medições disponíveis são limitados. Salientam também que o teor do defeito do artefacto investigado e a eficácia das actividades realizadas pela equipa responsável pela qualidade, são específicas por contexto e afectadas por vários factores influentes. Os factores influentes para as actividades efectuadas pela equipa responsável pela qualidade são por exemplo: a terminologia consistente da documentação, tipo da linguagem, o envolvimento do cliente no desenvolvimento do produto, entre outros. Os factores influentes no teor do defeito são por exemplo: a complexidade do projecto, a experiência do programador, o stress do programador, entre outros. Para responder a estes problemas, Kläs et *al.* propõem um método de previsão de defeitos híbrido HDCE (*Hybrid Defect Content and Effectiveness*) denominado de HyDEEP (*Hybrid Defect content and Effectiveness Early Predicition*). O HDCE combina os dados históricos do projecto disponíveis e a opinião de peritos, agrupando-os num modelo quantitativo causal reutilizável para factores que influenciam o teor do defeito e sua eficácia. Este modelo híbrido permite a construção de modelos específicos por contexto, que consideram a característica com maior relevância no contexto que origina o defeito [39].

Caldeira descreve uma experiência realizada sobre uma amostra de relatórios de incidentes, registados durante a operação de várias centenas de produtos de software comercial, durante um período de três anos (2005-2007), em seis países da Europa e América Latina [16]. Os incidentes foram introduzidos por clientes de um grande fornecedor de software no seu sistema de gestão de incidentes. O principal objectivo do processo de gestão de incidentes é restaurar o mais rápido possível o funcionamento normal do serviço e minimizar assim o impacto negativo sobre as operações de negócio, assegurando assim que os melhores níveis de qualidade de serviço e disponibilidade sejam mantidos. Como resultado disso, uma empresa de software pode fazer uso de um bom processo de gestão de incidentes para melhorar diversas áreas da sua actividade, em particular o desenvolvimento do produto, suporte ao produto, a relação com seus clientes e o seu posicionamento no mercado. Caldeira procurou identificar os factores que influenciam o ciclo de gestão de incidentes, bem como a detecção de padrões e/ou tendências na criação e resolução de incidentes com base numa abordagem de séries temporais. Além disso, Caldeira apresenta uma estimativa, avaliação e validação de diversos modelos ARIMA criados, com o propósito de efectuar previsões sobre a resolução de incidentes com base nos dados históricos da criação de incidentes. Por fim, Caldeira refere que, compreender relações-causais e padrões de gestão de incidentes pode ajudar as organizações que desenvolve software na optimização nos seus processos de apoio e alocação dos recursos adequados: pessoas e

orçamento.

De entre os trabalhos referidos nesta secção, o trabalho relatado por Caldeira em [16] é o mais próximo do presente estudo. A experiência nele relatada difere da nossa em vários aspectos, nomeadamente:

- No objecto de estudo, são analisados os relatórios de incidentes de produtos de software comercial, enquanto no presente é realizado um estudo sobre os relatórios de defeitos de um software *open source*.
- Na dimensão das amostras, são analisados 3 anos da actividade dos produtos, enquanto no presente é realizado a análise aos quase 10 anos de existência do software Eclipse.

No que respeita aos objectivos de pesquisa, estes são bastante próximos, pelo que esta experiência é uma excelente base de comparação para eventual conclusão de mecanismos e políticas que são seguidas nas empresas comerciais, também o são em projectos *open source*, funcionando assim como um mecanismo de validação cruzada mútua dos resultados obtidos.

Em relação aos trabalhos realizados por Wermelinger et *al.* [81] e Mens et *al.* [59] são usadas séries temporais do Eclipse com a finalidade de efectuar uma análise da sua evolução, não tendo sido a previsão um objectivo. Raja et *al.* [68] usam modelos ARIMA para gerarem previsões do número de defeitos de vários projectos *open source* de dimensões inferiores ao Eclipse. Em [39], Kläs et *al.* efectuam previsões baseadas num número pequeno de dados históricos e para colmatar essa falta, associam essa informação com a opinião de peritos de forma a que as previsões sejam o mais eficiente possível. Bernstein et *al.* utilizam modelos de previsão em [11], contudo o principal objectivo foi demonstrar que a inserção de informação sobre características temporais melhora significativamente o desempenho dos modelos de previsão.

6.3 Planeamento Experimental

6.3.1 Objectivos

Os objectivos de investigação propostos na sub-secção 6.1.3 são aqui redefinidos como objectivos experimentais. De referir que o contexto nunca é alterado ao longo do objectivos, ou seja, é sempre um estudo observacional sobre os defeitos do Eclipse disponíveis através do Bugzilla, e por isso é substituído por "(...)". De notar que o objectivo experimental **OE1** corresponde à questão de investigação **QI1**, e assim sucessivamente.

OE1: É nosso objectivo analisar os defeitos do Eclipse com a finalidade de caracterizar a sua evolução ao longo do tempo, procurando padrões sazonais, de modo a permitir a criação de modelos de previsão para que possam ser utilizados pelos gestores de projectos, "(...)"

OE2: É nosso objectivo analisar os defeitos do Eclipse com a finalidade de caracterizar a sua evolução ao longo do tempo, procurando tendências, de modo a permitir a criação de modelos de previsão para que possam ser utilizados pelos gestores de projectos, "(...)"

OE3: É nosso objectivo definir um modelo ARIMA para a série temporal do número de defeitos do Eclipse e efectuar previsões fiáveis, com a finalidade de credibilizar esta técnica, para que possam ser utilizados pelos gestores de projectos, "(...)"

OE4: É nosso objectivo verificar se as previsões do modelo ARIMA gerado, são de facto precisas e de confiança, com a finalidade de credibilizar os resultados obtidos bem como a técnica *per si*, para que possam ser utilizados pelos gestores de projectos, "(...)"

OE5: É nosso objectivo analisar a duração da resolução dos defeitos do Eclipse com a finalidade de caracterizar a sua evolução, procurando padrões sazonais, de modo a melhor compreender a evolução dessa propriedade ao longo do tempo, "(...)"

OE6: É nosso objectivo analisar a duração da resolução dos defeitos do Eclipse com a finalidade de caracterizar a sua evolução, procurando tendências, de modo a melhor compreender a evolução dessa propriedade ao longo do tempo, "(...)"

OE7: É nosso objectivo verificar se a eficiência do modelo de previsão ARIMA varia à medida que são retirados os dados históricos mais antigos e adicionados dados históricos mais recentes, com o intuito de permitir a caracterização dos modelos de previsão ARIMA para que possam ser utilizados pelos gestores de projectos, "(...)"

6.3.2 Unidades Experimentais

Na nossa experiência, a unidade experimental utilizada é o relatório de defeito do Bugzilla, que foi discutido na secção 4.1.2 e apresentado na figura 4.2 da mesma secção.

6.3.3 Material Experimental

O material experimental utilizado neste estudo compreende o conjunto de relatórios de defeitos recolhidos do Bugzilla até às 14h30, GMT, do dia 08-04-2010. Deste modo, a colecção de dados de defeitos inclui todos os defeitos registados desde o início do projecto Eclipse, à data referida acima.

6.3.4 Tarefas

A listagem que se segue pretende, de um modo simples e conciso, a exposição das tarefas necessárias para a realização do presente estudo. As tarefas são também exemplificadas no diagrama de actividades apresentado na figura 6.3. O diagrama de actividade segue a notação do UML 2.0 (*Unified Modeling Language*) no que concerne à forma de representação das actividades e dos dados consumidos na realização das actividades.

1. Recolha de informação do sistema de rastreio de defeitos Bugzilla através de um crawler já elaborado.
2. Modelação da informação e sua preparação para inserção na base de dados.
3. Inserção da informação na base de dados.
4. Transferência da base de dados para a ferramenta de análise estatística.
5. Desenvolvimento do modelo de séries temporais para o estudo da evolução.
6. Treino do modelo com dados históricos.
7. Validação do modelo.
8. Análise dos resultados.

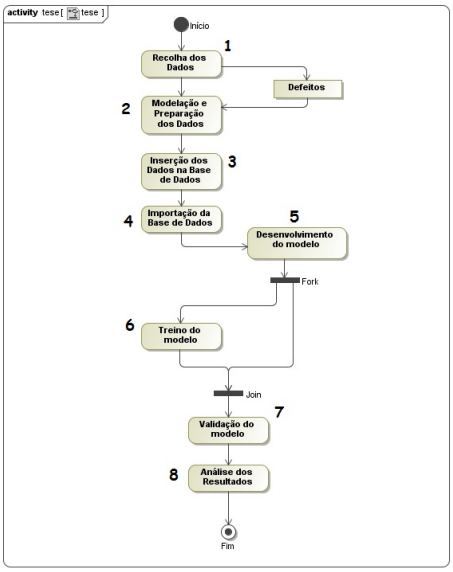


Figura 6.3 Diagrama de actividades das tarefas em UML.

6.3.5 Hipóteses e Variáveis

As variáveis usadas nesta experiência são descritas na tabela 6.1.

Tabela 6.1 Variáveis e tipos de Escala usadas na experiência.

Variável	Escala	Descrição
crea ts filter	Data	Data de criação do registo de defeito.
delta ts filter	Data	Data da última acção efectuada sobre o defeito.
priority num	Numérico	Identificador da prioridade do defeito.
classification num	Numérico	Identificador da classificação do defeito.
version num	Numérico	Identificador da versão do eclipse que pertence o defeito.
resolution num	Numérico	Identificador da resolução do defeito.
bug sev num	Numérico	Identificador da severidade do defeito.
bug stat num	Numérico	Identificador do estado do defeito.
component num eclipse	Numérico	Identificador do componente pertencente ao eclipse.
Month	Numérico	Valor que representa o mês em que o defeito foi criado.
Year	Numérico	Valor que representa o ano em que o defeito foi criado.
MonthYear	Data	Data que exprima o mês e ano da criação do defeito.
NDay	Numérico	Valor do dia mensal da criação do defeito.
WDay	Numérico	Valor do dia semanal da criação do defeito.
NYDay	Numérico	Valor do dia anual da criação do defeito.
Week	Numérico	Valor da semana da criação do defeito.
MonthDayYear	Data	Data que exprima o mês, dia e ano da criação do defeito.
N ALLBUG	Numérico	Número total de defeitos.
N ECLIPSE	Numérico	Número total de defeitos com a identificação eclipse.
duration solving	Numérico	Número que exprime a duração para se resolver um defeito.

Estas variáveis foram importantes para perceber o estado da base de dados e consequentemente o estado do repositório de defeitos Bugzilla. As variáveis com carácter temporal foram usadas para agregar a informação necessária à criação das séries temporais dependendo da granularidade escolhida. As variáveis que são identificadores, foram utilizadas para analisar as frequências de cada uma, percebendo assim como estão dispostas na base de dados. Por fim, estão as variáveis que serão utilizadas na execução das experiências. O número total de defeitos será utilizado para a criação dos modelos de previsão. O número total de defeitos com a identificação Eclipse, é usado para comparar a sua variação ao longo do tempo com o número total de defeitos. O valor que exprime a duração de resolução dos defeitos, é usado para verificar a sua variação ao longo do tempo.

Com base nos objectivos anteriormente referidos, são formuladas sete hipóteses, em que uma hipótese H_n corresponde à questão de investigação QI_n. É pretendido com cada hipótese, avaliar se a série temporal usada apresenta padrões sazonais ou uma tendência; se a metodologia ARIMA é uma aproximação válida para prever o número de defeitos, e se essas previsões são precisas. Pretendemos também avaliar se a resolução de defeitos ao longo do tempo apresenta algum padrão sazonal ou uma tendência e, por fim, verificar se a eficiência dos modelos ARIMA depende inteiramente dos pontos de dados.

$H1_0$: A série temporal do número de defeitos apresenta padrões sazonais.
$H1_1$: A série temporal do número de defeitos não apresenta padrões sazonais.

$H2_0$: A série temporal do número de defeitos apresenta uma tendência.
$H2_1$: A série temporal do número de defeitos não apresenta uma tendência.

$H3_0$: Prever o número de defeitos com modelos ARIMA é uma aproximação válida.
$H3_1$: Prever o número de defeitos com modelos ARIMA não é uma aproximação válida.

$H4_0$: As previsões do número de defeitos feitas com modelos ARIMA são precisas.
$H4_1$: As previsões do número de defeitos feitas com modelos ARIMA não são precisas.

$H5_0$: A resolução dos defeitos ao longo do tempo apresenta padrões sazonais.
$H5_1$: A resolução dos defeitos ao longo do tempo não apresenta padrões sazonais.

$H6_0$: A resolução dos defeitos ao longo do tempo apresenta uma tendência.
$H6_1$: A resolução dos defeitos ao longo do tempo não apresenta uma tendência.

$H7_0$: A eficiência dos modelos ARIMA depende do ponto inicial da janela de dimensão fixa que contém os pontos de dados da história do software.
$H7_1$: A eficiência dos modelos ARIMA não depende do ponto inicial da janela de dimensão fixa que contém os pontos de dados da história do software.

6.3.6 Desenho

O desenho utilizado nesta experiência segue o desenho de um estudo longitudinal, em que as observações são contagens do número de defeitos, realizadas periodicamente. O período para cada experiência varia. Nas experiências onde são criados modelos de previsão, existe um período de apreciação do modelo que é efectuado ao longo de 86 observações e um período de previsão de 15 observações na primeira experiência, e na experiência chamada de janela deslizante, o período de apreciação é feita ao longo de 74 observações, e o período de previsão ao longo de 12 observações, percorrendo 16 iterações, sendo esta a finalidade da janela deslizante.

No estudo efectuado sobre uma variável, são utilizadas todas as observações existentes.
A experiência segue o seguinte desenho:

O O x O O O X O O O x O O O O

Em desenho experimental o **O** representa as observações e o **X** representa os tratamentos ou programas. No nosso caso, correspondem à disponibilização de novas versões, que são, essencialmente, o factor que pretendemos demonstrar que influencia nos efeitos de sazonalidade observados. De salientar que o **X** corresponde a uma nova versão, por exemplo, a versão 3.0, ou a 3.1 e o **x** representa as *milestones*, por exemplo, a versão 3.1.1, ou a 3.1.2. De salientar que o desenho apresentado, corresponde a um ano completo, havendo por isso 12 observações e 3 tratamentos que correspondem a dois lançamentos de *milestones* e um lançamento de uma nova versão. O desenho está adaptado ao comportamento apresentado a partir da versão 3.0, em que existe o lançamento de uma *milestone* entre os meses de Fevereiro e Março. O lançamento da nova versão ocorre entre Maio e Junho, sendo entre Outubro e Novembro lançada uma nova *milestone*. Por esta razão o desenho apresentado nem sempre é igual mas que corresponde a um desenho típico para um ano.

6.3.7 Procedimento

Para proceder à realização desta experiência foi necessário fazer a recolha dos relatórios de defeitos do Eclipse. No projecto Eclipse, o rastreio de defeitos é suportado pelo Bugzilla.

As ferramentas que deram suporte a este estudo foram o **BugDownloader**, **MySQL** [2] e **Navicat Premium** [3] como suporte para a base de dados onde são armazenados os relatórios de defeitos e o **SPSS** [4] para a realização das análises estatísticas da mesma.

O **BugDownloader** é um *crawler* que permite a extracção dos relatórios de defeitos do Eclipse guardados no Bugzilla para ficheiros XML. Esta ferramenta foi elaborada por Alexis Descré e Fernando Brito e Abreu. O recurso ao BugDownloader deve-se ao facto de o Bugzilla não permitir a extracção directa de todos os relatórios de defeitos do Eclipse, tendo sido por isso necessário recorrer a este *crawler* que permite a execução de pedidos sucessivos ao Bugzilla.

O **MySQL** é um sistema de gestão de bases de dados *open source*, e o **Navicat Premium** é uma ferramenta que permite uma gestão fácil da base de dados. Esta ferramenta, para além

[2] http://www.mysql.com/
[3] http://www.navicat.com/
[4] http://www.spss.com/

de disponibilizar operações necessárias para a criação e elaboração da base de dados do tipo inserção dos relatórios de defeitos, extracção da informação dos ficheiros XML, e a exportação da informação para ficheiros de texto, permite ainda um fácil manuseamento de um número elevado de defeitos. Neste trabalho, foi usada uma versão experimental totalmente funcional.

Por fim, a ferramenta utilizada para realização das análises estatísticas, incluindo a criação de modelos de séries temporais, foi a *Statistical Package for the Social Sciences*, mais conhecida por **SPSS**.

O diagrama de sequência em UML 2.0 (*Unified Modeling Language*) da figura 6.4 apresenta, de uma forma clara e compacta, todas as actividades e procedimentos necessários para a realização deste estudo.

Neste diagrama podemos verificar a presença de todas as ferramentas mencionadas anteriormente, o resultado de algumas operações levam à criação de dados, como por exemplo, ficheiros de XML "XMLs", ficheiros de texto "TXTs" ou bases de dados "BD Auxiliar" e "BD Final". A comunicação entre os vários intervenientes é representada por simples trocas de mensagens, em que o nome ilustra o que é feito nessa comunicação. O actor representa o investigador que é responsável pela recolha dos dados e configurações necessárias para a boa execução desta experiência. Neste diagrama encontra-se também representada uma numeração de passos importantes efectuados ao longo da experiência. Cada passo será discutido um pouco mais em pormenor na descrição que se segue.

No ponto 1 o investigador necessita de configurar o BugDownloader para extrair os relatórios de defeitos, mais precisamente, necessita de escolher o intervalo de relatórios de defeitos que vai extrair naquela sessão. A extracção dos relatórios de defeitos é um processo demorado, daí ser aconselhado a escolha de intervalos não muito reduzidos mas também não muito grandes. Nesta experiência o intervalo para a extracção dos defeitos foi de 5000 defeitos. Este procedimento foi repetido até às 14h30, GMT, do dia 08-04-2010 e é representado no ponto 2. Deste modo, a base de dados de defeitos inclui todos os defeitos registados desde o início do projecto Eclipse, até ao momento acima referido. O último defeito extraído é identificado pelo id 308470, no entanto o total de defeitos extraídos foi 287176. Isto deve-se ao facto de existirem intervalos que não possuem defeitos, ou nos quais os defeitos foram já removidos. A cada extracção de relatórios é criado um ficheiro XML com a informação desse relatório e armazenado em disco.

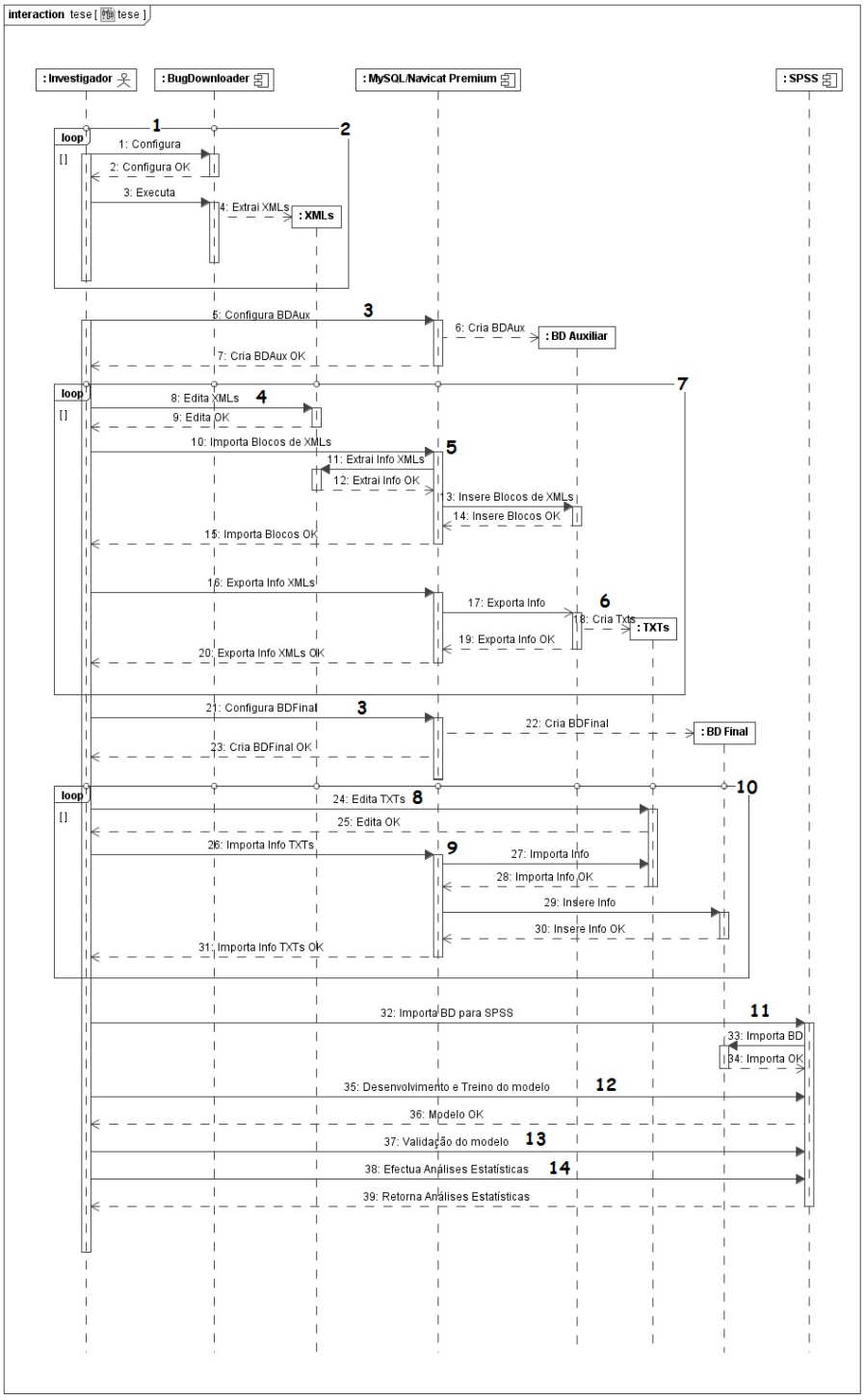


Figura 6.4 Diagrama de sequência das actividades em UML.

O ponto 3 aparece repetido no diagrama devido à necessidade de criar duas bases de dados: uma auxiliar para depósito temporário de defeitos, de forma a permitir a exportação para os ficheiros de texto, e outra final que conterá todos os defeitos. Para a criação de uma base de dados é necessária uma configuração prévia, mais específica na base de dados final que na auxiliar, visto a auxiliar servir apenas de repositório temporário.

Na fase seguinte desta experiência foi necessário editar alguns ficheiros XML, como descrito no ponto 4. Esta edição foi realizada com a finalidade de todos os relatórios seguirem a estrutura da base de dados final configurada com a estrutura representada na figura 6.5. Esta base de dados consiste numa tabela contendo todos os defeitos e sua informação, sendo a chave primária "bug id". Como nesta experiência foram tratados 2500 defeitos de cada vez, por razões de eficiência, foi necessário editar o primeiro defeito de cada bloco de 2500 defeitos, de modo que a ferramenta Navicat Premium assuma a estrutura dessa informação e possa repeti-la para os defeitos seguintes. Em consequência da estratégia adoptada, nalguns defeitos foram removidas algumas "tags" ou atributos que não entram na estrutura definida da base de dados. Estas "tags" ou atributos são, por exemplo, "thetext", "cc", "attachment" e todos os seus atributos associados ("attachid", "date", "desc", entre outros), "status whiteboard" e "dup id". No ponto 5 o investigador importa os ficheiros XML para a base de dados auxiliar, efectuando de seguida a exportação da informação de todos os ficheiros XML importados para um ficheiro de texto (ponto 6 do diagrama). O ponto 7 do diagrama assinala a repetição deste procedimento até não existirem mais relatórios a serem tratados.

Antes da inserção para a base de dados final, é necessário editar os ficheiros de texto gerados do procedimento anterior. Esta edição, representada no ponto 8 do diagrama, consiste na inserção do cabeçalho da tabela da base de dados final na primeira linha do ficheiro de texto. Esta edição é essencial para que a ferramenta Navicat Premium possa mapear correctamente a informação proveniente dos ficheiros de texto para a tabela da base de dados final. Nesta edição é também efectuada uma junção de dois ficheiros de texto de cada vez, isto é, dois ficheiros de texto, um contendo os defeitos de 1 a 2500, e outro contendo os defeitos de 2501 até 5000. O resultado desta junção é um ficheiro de texto com os defeitos de 1 a 5000. Este procedimento é feito até não existirem mais ficheiros de texto. Estas junções foram efectuadas com o principal objectivo de que o processo de inserção na base de dados final fosse mais eficiente. A fase seguinte é bastante simples e rápida, consistindo na importação dos ficheiros de texto para a base de dados final, representada no ponto 9, sendo esta operação repetida até não existirem

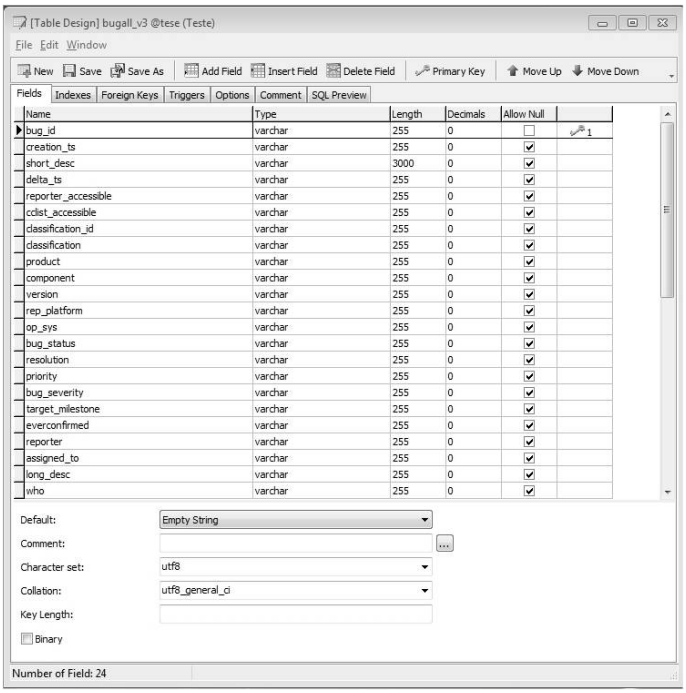


Figura 6.5 Estrutura da base de dados dos relatórios de defeitos.

mais ficheiros de texto para serem inseridos, como ilustrado no ponto 10. No final deste processo, para todos os defeitos extraídos do Bugzilla, verificou-se que o número total de defeitos inseridos na base de dados foi de 287162, o que comparando com o número total de defeitos extraídos, 287176, mostra que 14 defeitos se perderam no processo de inserção. Esta perda de informação deverá ter ocorrido no processo de importação dos dados para a base de dados, mas tem um efeito negligenciável no estudo que pretendemos realizar.

Por fim, o ponto 11 representa a operação de importação da base de dados para o SPSS, e após essa importação ter sido feita, poderemos então, de acordo com o ponto 12, proceder ao desenvolvimento do modelo e do seu treino com dados históricos. Após o treino do modelo, é efectuado a validação do mesmo (ponto 13), que produzirá resultados que serão depois analisados estatisticamente (ponto 14).

Na tabela 6.2, é feita a correspondência, entre as tarefas do diagrama de actividades, apresentado na figura 6.3, e o procedimento necessário para a realização desta experiência apresentado no diagrama de sequência da figura 6.4, ambos identificados pelo correspondente número nas respectivas figuras.

Tabela 6.2 Correspondência entre o Diagrama de Actividades e o Diagrama de Sequência.

Diagramas	
Actividade	Sequência
1	1, 2
2	3, 4, 5, 6, 7, 8
3	9, 10
4	11
5	12
6	12
7	13
8	14

6.3.8 Procedimento de Análise

Inicialmente é feita uma análise sobre gráficos chamados de correlogramas para identificar na série temporal possíveis padrões sazonais ou tendências. Estes gráficos mostram a correlação entre o número de defeitos em sucessivos intervalos de tempo. Os correlogramas também são utilizados para ajudar a identificar os parâmetros que compõem os modelos de previsão ARIMA. Os modelos são analisados através de umas estatísticas de ajuste e de erro. A estatística de ajuste indica-nos se o nosso modelo está ajustado à série temporal, enquanto que a estatística de erro, indica-nos as percentagens de erro do nosso modelo. Um modelo é bom quando é apenas preenchido por ruído branco. Podemos verificar essa propriedade com o teste de Ljung Box. Quanto maior a estatística Sig, melhor, indicando-nos assim que o modelo escolhido é apropriado e ajusta-se bem à série. Esta estatística juntamente com as outras mencionadas serão comparadas entre modelos para podermos fazer a distinção de qual o modelo é o mais fiável e de confiança.

Para saber mais sobre correlogramas, sobre o processo de identificação de parâmetros, ou sobre os modelos ARIMA e que propriedades são necessárias verificar para poder usar tais modelos, que métricas de ajuste e de erro são usadas, sugerimos a consulta do anexo D com informação mais detalhada.

6.4 Execução

6.4.1 Amostra

Da base de dados criada e preenchida com todos os defeitos extraídos do Bugzilla, nem todos os atributos foram utilizados para a realização deste estudo. Alguns atributos da base de dados não representam informação relevante, já que eram todos preenchidos pelo mesmo valor. No caso de *reporter accessible*, *cclist accessible* e *everconfirmed* todas as linhas estavam preenchidas com o valor 1. Estes atributos não têm qualquer capacidade discriminante entre os registos de defeitos, podendo portanto ser descartados das análises a realizar. Os atributos *short desc* e *long desc* também não foram considerados para a realização de análises estatísticas visto se tratarem de descrições curta e longa, respectivamente, do defeito em causa. O atributo *bug id* também não foi analisado estatisticamente, por se tratar de uma chave única para cada defeito.

Depois da exclusão destes atributos passou-se então à preparação de dados da base de dados para permitir a análise estatística da mesma. Como a base de dados foi importada do Navicat Premium através do *Wizard* do SPSS, a atribuição dos tipos de dados de cada atributo foi automática, sendo que o tipo de dados por omissão é o tipo *String*, que não é o mais conveniente em vários casos. Foi, portanto, necessário proceder à correcção dos tipos de dados. Em particular, transformou-se o *creation ts* e *delta ts* em dados do tipo *Date*, tendo sido criada uma variável denominada *duration solving* que devolve o tempo em dias que levou o respectivo defeito a ser resolvido.

Criámos também variáveis para a prioridade do defeito que varia entre 1 e 5, etiquetadas respectivamente: P1, P2, P3, P4 e P5, ordenadas da prioridade mais urgente para menos urgente. Foi criada também uma variável de classificação que varia entre 1 e 13 em que para cada valor é atribuído uma etiqueta que identifica a que classificação aquele defeito pertence, ou seja, o 1 corresponde a Eclipse Foundation, 2 corresponde ao Eclipse, entre outros. Por exemplo, neste caso a codificação em inteiros não pode ser usada para nada que não a discriminação entre grupos, ou seja, o número de cada grupo é completamente irrelevante, para tudo o que não seja distinguir entre as várias classificações. Todos os campos que caracterizam o estado, resolução e severidade do defeito, foram também normalizados com a criação de novas variáveis. Estas normalizações podem ser visualizadas na tabela 6.3:

Tabela 6.3 Recodificações efectuadas à variáveis da base de dados.

Valor	Classification id	priority num	bug sev num	bug stat num	resolution num
	Nominal	Ordinal	Nominal	Nominal	Nominal
1	Eclipse Foundation	P1	blocker	NEW	Blank
2	Eclipse	P2	critical	ASSIGNED	"—"
3	BIRT	P3	major	REOPENED	DUPLICATE
4	Tools	P4	normal	RESOLVED	FIXED
5	Technology	P5	minor	VERIFIED	INVALID
6	TPTP		trivial	CLOSED	LATER
7	DSDP		enhancement		NOT ECLIPSE
8	WebTools				REMIND
9	DataTools				WONTFIX
10	STP				WORKSFORME
11	Modeling				
12	RT				
13	SOA				

Para facilitar a realização de algumas perguntas sobre o Eclipse em si, foi criada uma variável ordinal denominada de *version num* que retorna, como nos casos anteriores, uma lista de 1 a 30 das versões do Eclipse que estão presentes na base de dados, ou seja, o valor 1 corresponde a versão 0.9, a 2 = a versão 1.0, 3 = 2.0, assim sucessivamente. De salientar que, embora existam defeitos na amostra associados a uma versão 0.9, não é conhecida nenhuma versão 0.9 do Eclipse. Isto leva-nos a considerar que os defeitos associados à versão 0.9 serão referentes a algum *plug-in* que por sua vez se liga ao Eclipse, mas que erradamente foi classificado com o *classification id* do Eclipse. Para tal, também foi necessário criar um filtro, que apenas capturasse os defeitos que tivessem o *classification id* = 2 que corresponde ao Eclipse em si como mostra a tabela 6.3, e excluísse os defeitos com *classification id* diferentes de 2.

A filtragem dos defeitos varia conforme as experiências realizadas. Por exemplo, em análises estatísticas sobre o estado da base de dados, a amostra é a base de dados completa. Algumas análises direccionadas para as versões do eclipse, são filtradas através do atributo *classification id* = 2 que corresponde ao Eclipse em concreto. Para a criação dos modelos de séries temporais, foi necessário proceder a operação de agregação e junção de forma a podermos criar um conjunto de dados ordenados por data (mês e ano) com o número de defeitos em cada mês.

De salientar também que, na criação dos modelos de previsão e análises descritivas feitas à base de dados, o mês de Outubro de 2001 e o mês de Abril de 2010 não foram considerados,

com a finalidade de evitar influências espúrias no comportamento da série temporal. Em Outubro de 2001 observa-se um número anormalmente elevado de defeitos registados, que deverão resultar da introdução inicial de defeitos até então coleccionados de outra forma, bem como da realização de testes ao sistema de rastreio de defeitos. Os dados de Abril de 2010 são ignorados dado que a recolha de dados foi efectuada apenas até ao dia 08-04-2010. Como nos modelos de previsão a granularidade usada foi do número de defeitos por mês, não faria sentido manter um valor muito inferior para um mês em resultado de uma recolha correspondente a apenas à primeira terça parte desse mês.

6.4.2 Recolha dos Dados Efectuada

A recolha dos dados foi efectuada até às 14h30, GMT, do dia 08-04-2010. Deste modo, a colecção de dados de defeitos inclui todos os defeitos registados no Bugzilla desde o início do projecto Eclipse, até ao momento acima referido.

6.5 Análise

6.5.1 Estatísticas Descritivas

Após a recodificação da base de dados, verificou-se que quase todos os atributos pertencentes à base de dados não possuíam um valor numérico mas sim um valor identificativo. A única variável com valor numérico foi a variável criada por nós, que contabiliza os dias necessários para a resolução de um defeito. As outras variáveis têm valores identificativos que são contabilizados com o intuito de identificar o número de defeitos que tem um determinado valor identificador. Muitas dessas análises podem ser consultadas em detalhe nas tabelas e respectivas figuras que se encontram no anexo A.

De seguida serão apresentados alguns gráficos que ilustram a variação de defeitos ao longo do tempo, desde o início da criação deste repositório de dados à data em que deixamos de extrair relatórios de defeitos.

A figura 6.6 mostra a variação de defeitos ao longo do tempo, incluindo quer os defeitos relativos apenas ao Eclipse, isto é, com o *classification id* = 2, quer o número total de defeitos que se encontram no repositório de defeitos. Após a análise deste gráfico podemos verificar que o número de defeitos relativos ao Eclipse tem vindo a diminuir comparado com o número total

de defeitos, isto porque, cada vez mais o Eclipse recorre a utilização de componentes externos ao seu *core* para desempenhar as funcionalidades propostas, justificando assim a arquitectura baseada em *plug-ins*.

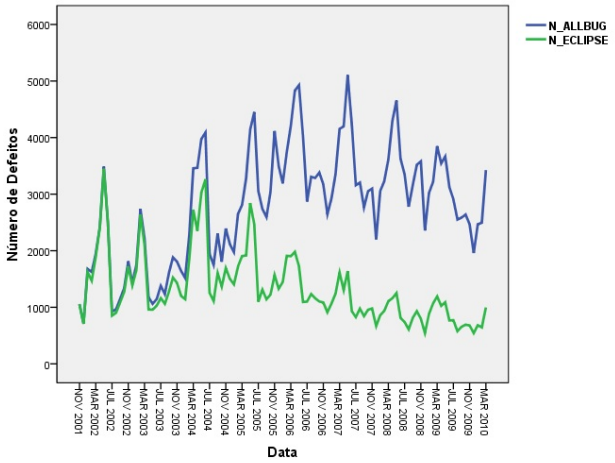


Figura 6.6 Gráfico de todos os defeitos (azul) Vs. os defeitos do Eclipse (verde).

As figuras seguintes representam a variação do número de defeitos relativos ao Eclipse, mas agora com a diferenciação por versões. Na figura 6.7 são discriminados os defeitos em função do tempo para todas as versões do Eclipse; na figura 6.8a, por apenas as versões principais; e na figura 6.8b por todas as versões intermédias ou *milestones*.

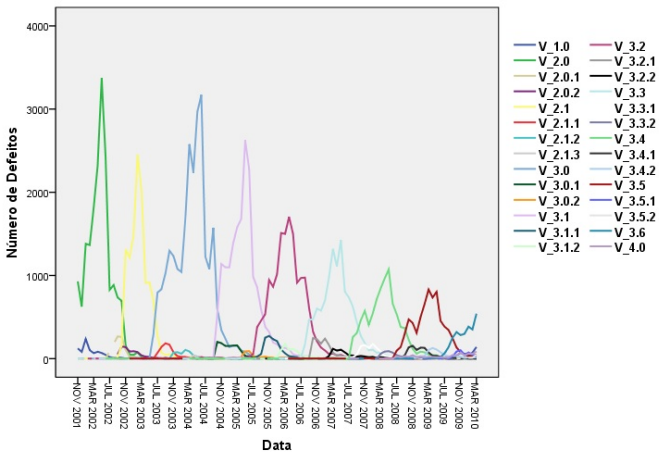
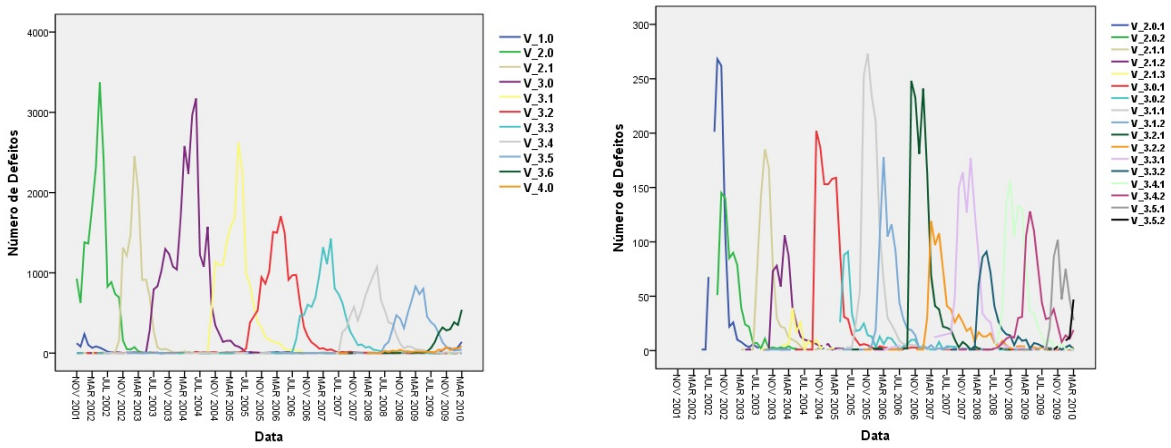


Figura 6.7 Gráfico de todas as versões do Eclipse ao longo do tempo.



(a) Gráfico das versões principais do Eclipse.

(b) Gráfico das *milestones* do Eclipse.

Figura 6.8 Gráficos das versões principais e *milestones* do Eclipse.

Após a análise destes gráficos, verificamos que os picos das versões principais ocorrem quase sempre na mesma altura, nos meses de Maio e Junho. Estes meses antecedem sempre o lançamento de uma nova versão com a excepção do ano de 2003, que corresponde à versão do Eclipse 2.1. Esta excepção deve-se ao facto da versão 2.1 do Eclipse ter sido a única com três versões intermédias lançadas, enquanto que as outras versões principais do Eclipse têm apenas duas versões intermédias lançadas entre duas versões principais. Mas desde as versões 3.0 até a actualidade tem-se verificado este comportamento sazonal de uma versão atingir o pico de defeitos nos meses de Maio e Junho anteriores ao lançamento de uma nova versão. Sobre os dados em função das *milestones* (figura 6.8b), verificamos que normalmente a primeira *milestones* tem sempre mais defeitos que as seguintes, o que transmite a ideia de uma boa capacidade de detecção e resolução de defeitos por parte dos membros pertencentes a equipa de desenvolvimento.

6.5.2 Redução do Conjunto de Dados

Nesta experiência não foi necessário recorrer à redução do conjunto de dados inválidos. De salientar que em todas as análises feitas à base de dados, e na criação das séries temporais, os defeitos contabilizados no mês de Outubro de 2001 e do mês de Abril de 2010 não foram considerados. Na criação dos modelos de previsão, as amostras foram divididas em duas partes, as amostras usadas pelo modelo para se ajustar à série temporal chamam-se de período de

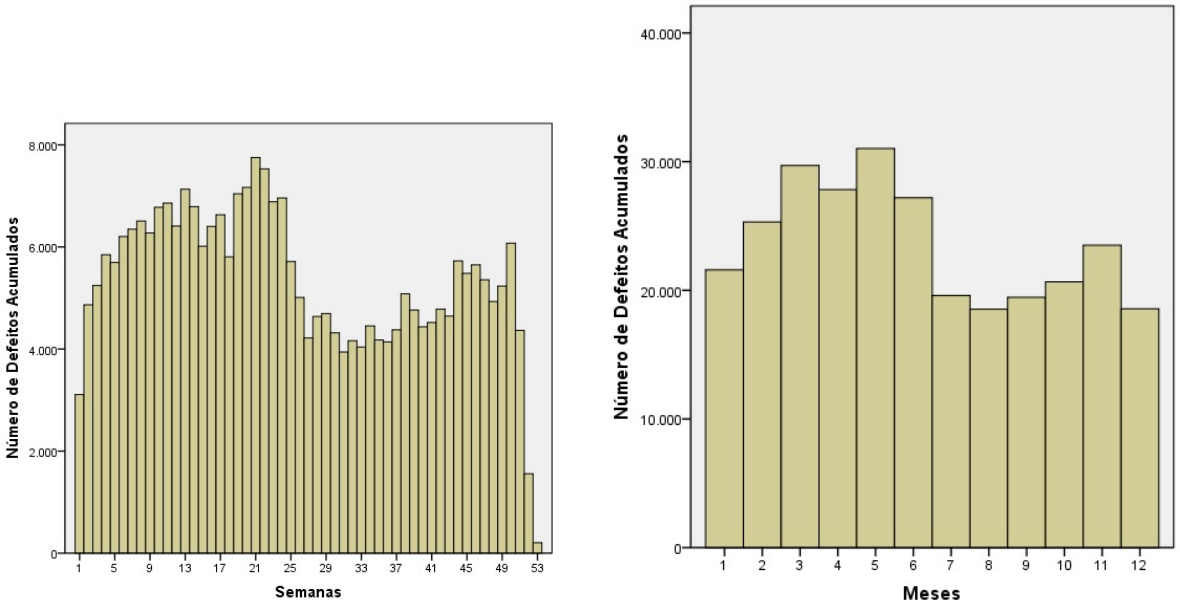
estimação, normalmente 86 meses, e a outra parte, o período de previsão, que é o intervalo das previsões efectuadas pelo modelo, normalmente 15 meses. Na experiência de janela deslizante que corresponde à **QI7**, essas dimensões foram alteradas; o período de estimação (PE) passou para 74 meses e o período de previsão (PP) passou para 12 meses. Esta experiência consiste na criação do modelo para aquele intervalo de tempo, após o qual é efectuada a deslocação da janela em 1 unidade, neste caso 1 mês. Para exemplificar na 1ª iteração - PE Nov2001-Dez2007, PP Jan2008-Dez2008, 2ª iteração - PE Dez2001-Jan2008, PP Fev2008-Jan2009, e assim sucessivamente até atingir o último mês, Março de 2010.

6.5.3 Identificação das Hipóteses e Testes

6.5.3.1 Padrões Sazonais

Um dos principais objectivos desta experiência é verificar a existência de padrões sazonais no processo de submissão de registos de defeitos no Bugzilla. Para descobrir possíveis padrões, resolvemos gerar várias análises sobre o número total de defeitos existentes na nossa base de dados, modificando a granularidade de cada análise. A granularidade dessas análises, passou por exemplo de dias (1-366), para dias de semana (1-7), dias de mês (1-31), para semanas no ano (1-53) e meses no ano (1-12). Contudo só vamos apresentar as análises feitas com a granularidade de semanas no ano, meses no ano e, no final, os dias de semana, porque as restantes não acrescentam informação para além daquela que é apresentada pelas granularidades escolhidas. Nos diagramas apresentados de seguida, o número de defeitos é disposto por duas granularidades, semanas no ano na figura 6.9a e meses no ano, na figura 6.9b. Informação detalhada sobre as frequências dos números de defeitos, é fornecida em anexo, mais precisamente na tabela A.3 para a granularidade semanas no ano e na tabela A.4 para os meses no ano.

Na figura 6.9, podemos verificar que na primeira metade do ano o registo de defeitos é mais intenso que na segunda metade do ano, tendo o seu pico no mês de Maio. Verificamos também que a seguir ao mês de Junho, o número de defeitos decresce e apenas torna a aumentar a partir de Setembro. Por fim, o número de defeitos registados decresce bruscamente em Dezembro. Optamos por apresentar dois diagramas com a mesma informação, mas com granularidades diferentes, para facilitar a percepção da variação do número de defeitos registados, e perceber se as quedas entre meses foram mais acentuadas, fornecendo-nos assim, a granularidade por semanas. A queda do final de ano, é uma queda mais acentuada na granularidade das semanas do que na dos meses. Isto deve-se essencialmente ao facto de os anos terem 52 semanas



(a) Número de defeitos por semanas em todos os anos. (b) Número de defeitos por meses em todos os anos.

Figura 6.9 Número de defeitos por semanas e meses em todos os anos.

completas, mas em que nem sempre a 52ª semana é completa, ou que, garantidamente, a 53ª, quando existe, será mais pequena. *Idem* para a primeira semana do ano, que é mais pequena. Isto porque as semanas são calculadas a partir do dia 1 do ano, que pode calhar em qualquer dia da semana. Estas peculiaridades, juntas aos fenómenos festivos fazem com que a média de defeitos registados seja ligeiramente inferior.

Na figura 6.10, é apresentado o gráfico que dispõe os números de defeitos registados em todos os anos, mas por dias semanais e que é apoiada pela tabela A.5 que se encontra em anexo. A maior parte dos defeitos foram registados em dias laborais. Poder-se-ia pensar que, por ser *open source* incluindo contribuições quer do *staff* de desenvolvimento do eclipse, quer de colaboradores voluntários alheios, faria prever um grande número de defeitos registado no fim de semana, no entanto não se verifica. Verificamos no diagrama da figura 6.10 que o acumulado do número de defeitos, cresce de segunda-feira(2) a quarta feira(4) e depois assume um comportamento decrescente até sexta-feira(6). Sábado(7) e domingo(1) apresentam uma actividade muito inferior aos restantes dias.

A análise dos componentes de sazonalidade e de tendência são extremamente importantes

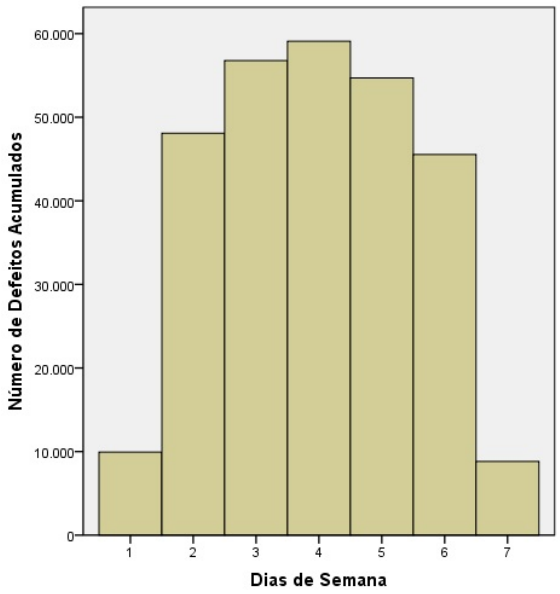
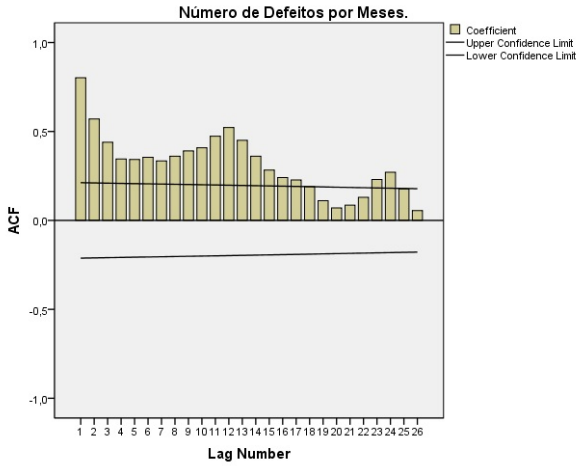


Figura 6.10 Número de defeitos por dias semanais em todos os anos.

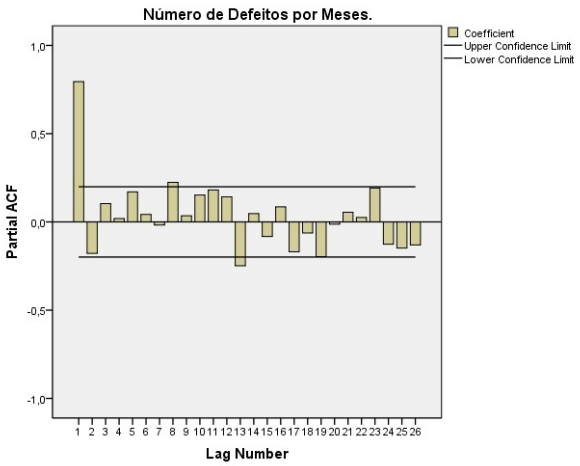
quando queremos modelar séries temporais. Os parâmetros ARIMA *(p,d,q)* representam os parâmetros não sazonais e ARIMA *(ps,ds,qs)* representam a parte sazonal do modelo. São obtidos através de uma observação cuidada dos padrões das respectivas séries temporais. Definições completas sobre os modelos ARIMA e os seus parâmetros podem ser encontrados em detalhe no anexo D.

Análise da sazonalidade (H1) Os padrões sazonais das séries temporais podem ser verificados através de correlogramas. Tanto a função de auto-correlação (ACF) como a função parcial de auto-correlação (PACF) podem garantir a presença de qualquer padrão. Como visto em [14], por Box et *al.*, a PACF é um meio mais preciso de analisar a sazonalidade. Mais informação sobre estas funções pode ser encontrada em detalhe no anexo D.

Na figura 6.11a, verificamos que existe uma variação em curva descendente, e onde o mais forte factor de correlação ocorre a cada 12 *lag* (intervalos de tempo), considerando o número de defeitos por mês. Como sabemos que cada observação na nossa série temporal representa um mês, é possível suspeitar a existência de um padrão sazonal anual. Para confirmação, é necessário analisar também o PACF. Após a análise do PACF, não é óbvia a existência de um padrão sazonal anual, visto que o esperado seria existir um pico no *lag* 12 e isso não acontece, o que não anula, nem contraria, a hipótese de existir um padrão sazonal anual. Simplesmente



(a) Função de Auto-correlação (ACF).

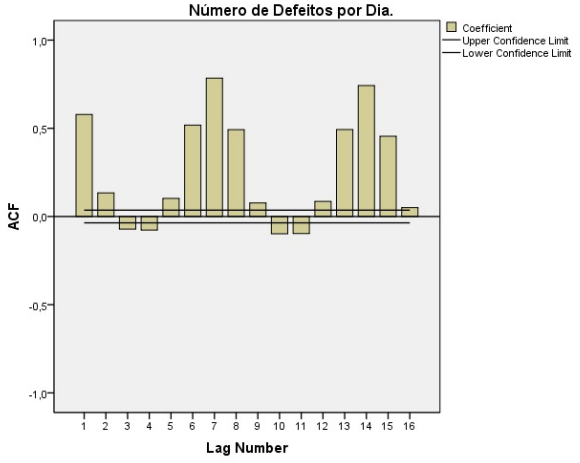


(b) Função parcial de Auto-correlação (PACF).

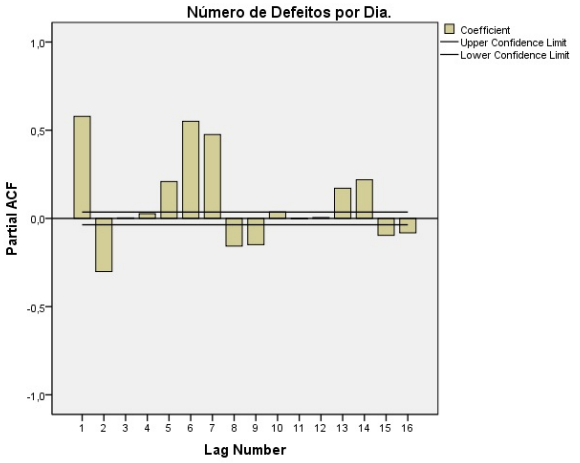
Figura 6.11 ACF e PACF da série temporal do número de defeitos por meses.

não valida para já a suspeita.

Em análise de outra série temporal e suas respectivas ACF e PACF, é também possível verificar a existência de algum padrão sazonal. Comparativamente com a série anterior, esta difere na granularidade temporal: nesta, a granularidade é em dias, na anterior, em meses.



(a) Função de Auto-correlação (ACF).



(b) Função parcial de Auto-correlação (PACF).

Figura 6.12 ACF e PACF da série temporal do número de defeitos por dias.

Como demonstra a figura 6.12a, o modelo apresenta um comportamento sinusoidal e o

mais forte factor de correlação ocorre a cada 7 *lags*. Como sabemos que cada observação na nossa série temporal representa um dia, é possível suspeitar um padrão sazonal semanal. Para confirmar, procede-se como anteriormente analisando-se o PACF. Embora com uma moderada correlação, mas bem acima dos intervalos de confiança, a PACF mostra exactamente o mesmo padrão a cada 7 *lags*, não deixando qualquer dúvida da existência do padrão semanal.

Como conclusão é possível suspeitar de um padrão sazonal anual na primeira série temporal (por meses) a ser confirmado com análises posteriores. A segunda série temporal (por dias) apresenta claramente um padrão sazonal semanal, que se repete a cada 7 dias.

Apesar da série temporal por dias ter apresentado e confirmado um padrão sazonal semanal, a série temporal em foco nesta experiência é a dos números de defeitos acumulados por meses, e consideramos que esta granularidade é suficiente para ser efectuada a nossa experiência com resultados satisfatórios.

Análise de Tendência (H2) Não existe uma maneira exacta de identificar os componentes influentes da tendência dos dados de uma série temporal. Contudo, se a tendência é monótona (aumentando ou diminuindo gradualmente) a identificação da tendência é mais simples. Tal pode ser verificado na série temporal representada na figura 6.13. Para analisar a tendência, é usada a série temporal de granularidade por meses, desenvolvida no presente estudo, representando o número de defeitos por meses (figura 6.13).

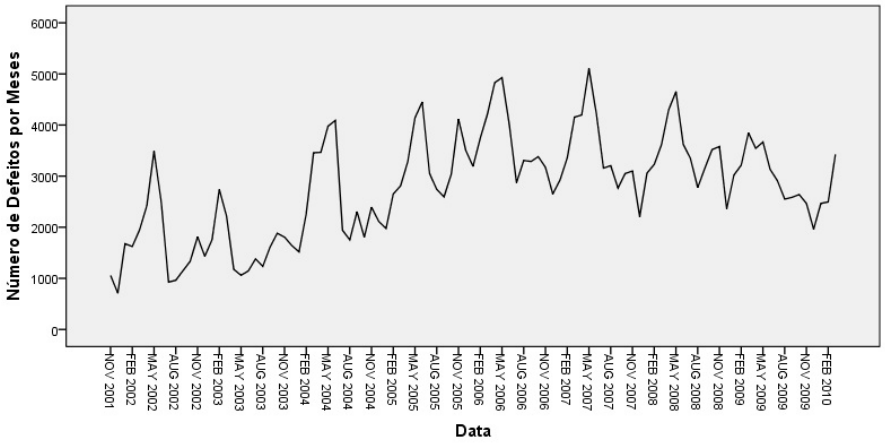


Figura 6.13 Série Temporal - Número de defeitos por meses.

Na série temporal apresentada na figura 6.13 os dados sugerem a existência de uma tendência com um padrão de crescimento muito subtil. Para confirmar esse comportamento procedeu-se a uma decomposição sazonal. O procedimento da decomposição sazonal, decompõe a série (i) num componente sazonal, (ii) num componente combinado de tendência e ciclo e (iii) num componente de erro. O procedimento da decomposição sazonal cria quatro novas séries [5]:

1. Uma série com os factores de ajustamento sazonal (SAF) que indicam o efeito de cada período no nível de cada série.

2. Uma série ajustada sazonalmente (SAS), que é uma nova série com os valores obtidos após a remoção da variação sazonal das séries originais.

3. Uma série com os componentes de suavização de tendência e ciclo (STC) que mostram a tendência e o comportamento cíclico presentes nas séries.

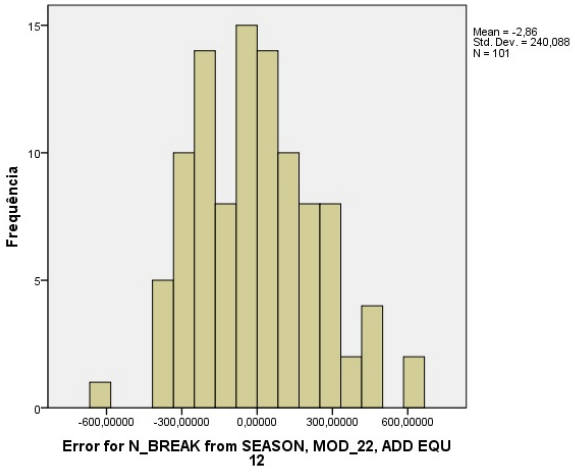
4. Uma série com os valores de resíduos ou erros (ERR) que representam os valores que sobram após a remoção das séries, dos componentes sazonais, de tendência e ciclo.

Figura 6.14 Série STC - Número de defeitos por meses com as variações sistemáticas sazonais removidas.

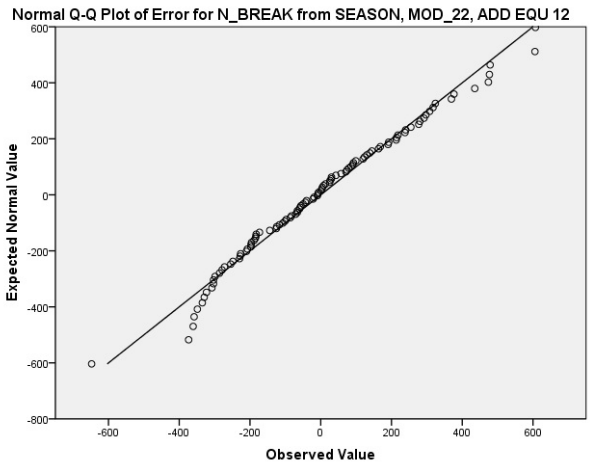
Na figura 6.14 é possível visualizar os componentes de nivelamento de tendência e ciclo, que resultaram da decomposição sazonal, que mostra uma tendência crescente. De salientar que os maiores crescimentos verificados ocorrem aquando do lançamento das versões 3.0 (Junho de

[5] O componente sazonal é decomposto em duas séries, SAF e SAS

2004) e 3.1 (Junho de 2005) do Eclipse, mas que após esses crescimentos, a série temporal mantém-se estacionária. Para poder afirmar com certeza sobre a tendência da série temporal principal, é necessário efectuar uma validação cuidadosa. Para tal, devem-se validar as séries *de-trended*, através da inspecção dos seus valores residuais ou erros (séries ERR). Os valores residuais resultam da diferença entre os valores do *output* dos modelos das séries *de-trended* previstas com o *output* medido das observações básicas da série. Estes resíduos representam a porção da variação que não é explicada pelo modelo. Os resíduos não devem ser correlacionados para as séries serem consideradas válidas. Resíduos não-correlacionados significam também que não existe nenhuma correlação entre as séries originais e as séries *de-trended*, e como tal deve ser verificado se o resíduos seguem uma distribuição normal através da análise do histograma e *QQ-plot* dos resíduos. No histograma uma distribuição normal tem forma de sino e tende para o eixo horizontal à medida que se afasta da média, no *QQ-plot* para verificar se os resíduos seguem uma distribuição normal, os pontos dispõem-se aproximadamente em linha recta.



(a) Histograma dos resíduos (Séries ERR).



(b) *QQ-plot* dos resíduos (Séries ERR).

Figura 6.15 Histograma e *QQ-plot* para a validação da série.

Observando o histograma apresentado na figura 6.15a e o diagrama da probabilidade normal da figura 6.15b, podemos verificar que a série dos resíduos (Séries ERR) que resultou da decomposição sazonal segue uma distribuição normal. Confirmada esta propriedade dos resíduos, é possível agora afirmar que a presente série temporal encontra-se nas condições adequadas para a aplicação dos modelos ARIMA.

6.5.3.2 Modelação da série temporal mensal com ARIMA

Introdução Os procedimentos de modelação e previsão requerem um conhecimento sobre o modelo matemático do processo. Na vida real, tanto na investigação como na prática, os padrões dos dados não são óbvios, observações individuais envolvem um erro considerável. É necessário não apenas descobrir os padrões ocultos nos dados mas também de gerar previsões. A metodologia ARIMA explicada em [14], permite-nos fazê-lo. Esta metodologia oferece um grande poder e flexibilidade, mas tem o seu grau de complexidade.

O modelo geral inclui um parâmetro *auto-regressivo* (AR) bem como um parâmetro *média móvel* (MA), e explicitamente inclui na formulação do modelo a diferenciação (ver secção 6.5.3.2). Especificamente, os três tipos de parâmetros no modelo são: os parâmetros auto-regressivos (p), o número de passos de diferenciação (d) e os parâmetros de média móvel (q). Nesta notação os modelos são resumidos como *ARIMA (p,d,q)*, por exemplo, um modelo descrito como (0,1,2) significa que contém 0 (zero) parâmetros auto-regressivos (p) e 2 parâmetros de média móvel (q) que foram calculados após a série ter sido diferenciada uma vez (d=1).

Identificação do modelo Os modelos ARIMA requerem 3 parâmetros não sazonais, *p,d,q* e 3 parâmetros sazonais, *ps,ds,qs*. Este modelo pode ser decomposto nos parâmetros não sazonais AR(p), diferenciação(d) e MA(q). Similarmente tem uma parte sazonal decomposta num parâmetro auto-regressivo sazonal AR(ps), numa diferenciação sazonal (ds) e num parâmetro de média móvel sazonal MA(qs). A estacionariedade não tem de existir originalmente numa série. Contudo, uma série tem de ser estacionária para que a aplicação dos modelos *ARIMA (p,d,q)(ps,ds,qs)* seja correcta. Uma série temporal diz-se estacionária quando não existem mudanças sistemáticas na média, não havendo tendência, se não existirem mudanças sistemáticas na variação e se as variações estritamente periódicas tiverem sido removidas.

Diferenciação Quando uma série temporal não é estacionária, é de prática comum submeter a série a uma transformação. Normalmente essa transformação, é a diferenciação. O número de vezes que a série necessita de ser diferenciada até se encontrar estacionária, é o valor que assume *d* nos modelos ARIMA. Mais informação sobre a diferenciação encontra-se em detalhe no anexo D

Como observado anteriormente na figura 6.13, é possível concluir que a presente série não é estacionária, devido a ter uma ligeira tendência crescente e variações estritamente periódicas. Para torná-la então estacionária, deve ser aplicada uma diferenciação à série. O resultado dessa

transformação é apresentado na figura 6.16.

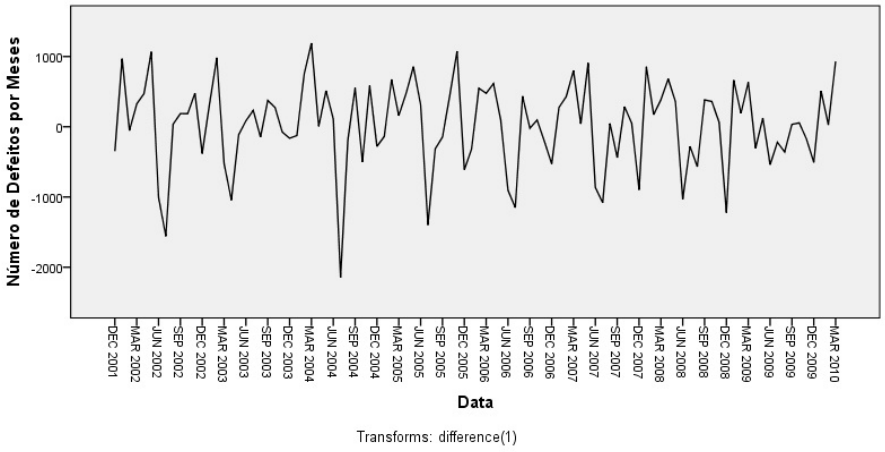


Figura 6.16 Série Temporal com diferenciação (1).

É também comum aplicar transformações antes da diferenciação com a finalidade de estabilizar a variação da série temporal. Usualmente são transformações logaritmo natural ou raiz quadrada. No presente caso não foi necessário aplicar nenhuma dessas transformações. Após diferenciar a série uma vez, esta tornou-se estacionária, sem nenhuma tendência e com as variações estritamente periódicas removidas.

A função parcial de auto-correlação (PACF) apresentada na figura 6.17, apoia a afirmação anterior de que a série se encontra já estacionária. Como é possível observar, os picos de correlação nos 12 primeiros intervalos de tempo, terminam abruptamente. Tal comportamento é o esperado para uma série temporal estacionária.

Como é apenas necessário diferenciar a série uma vez para torná-la estacionária, identifica-se assim o parâmetro *d=1*.

Parâmetros não sazonais Os parâmetros AR(p) e MA(q) podem ser obtidos através da análise da função de auto correlação (ACF) e da função parcial de auto correlação (PACF).

As funções apresentadas na figura 6.11, no estudo feito para a análise da sazonalidade, mostram-nos que a ACF da figura 6.11a não evidencia um padrão definido, e que o PACF da figura 6.11b tem um pico no intervalo de tempo 1 e sem correlação nos outros intervalos. Com isto é possível estimar o parâmetro auto regressivo AR(p)=1 e o parâmetro de média móvel MA(q)=0. Estas noções racionais para a identificação dos parâmetros p e q estão descritas em detalhe no anexo D.

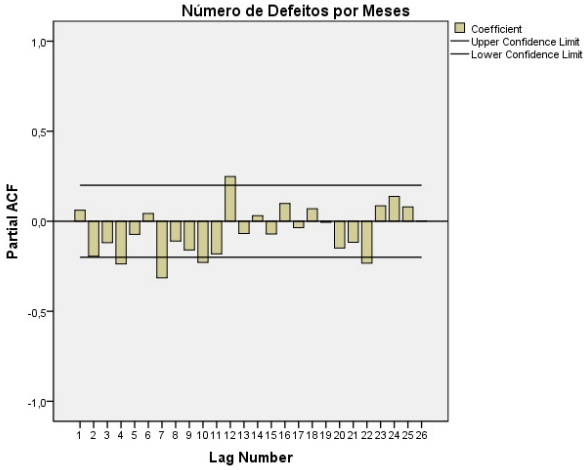
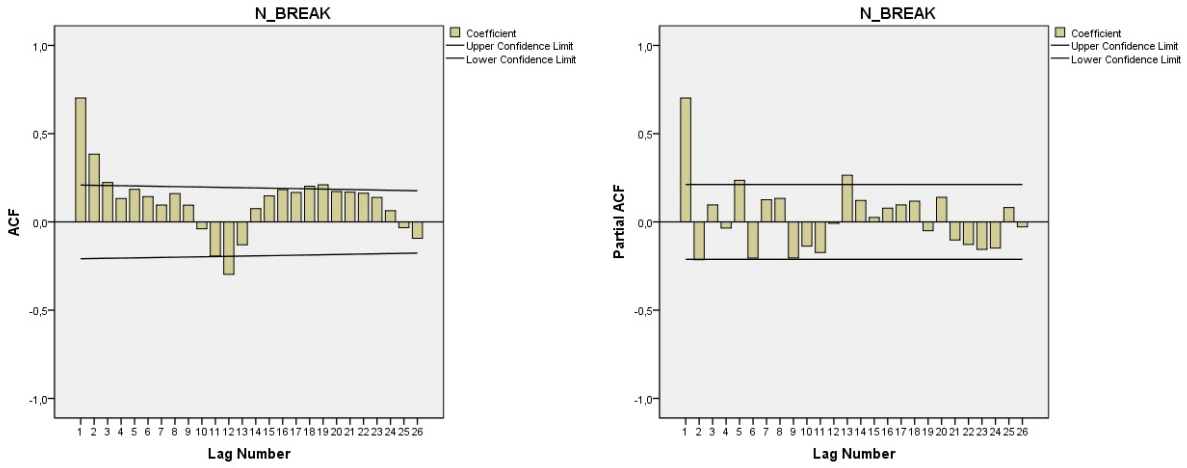


Figura 6.17 PACF da série temporal após diferenciação (1).

Parâmetros sazonais Como mencionado anteriormente, é suspeitado fortemente que a presente série temporal em estudo tenha um período sazonal anual (12 meses) e como tal, é necessário a aplicação da diferenciação sazonal uma vez. A ACF e PACF após a aplicação da diferenciação sazonal, possibilita estimar os valores para os parâmetros AR(ps) e MA(qs) do modelo apresentado.



(a) ACF após diferenciação sazonal (1).

(b) PACF após diferenciação sazonal (1).

Figura 6.18 ACF e PACF da série temporal após a diferenciação sazonal (1).

Após a análise da ACF e PACF, verifica-se que o PACF apresenta um pico no intervalo de

tempo 1 e sem correlação nos restantes intervalos (figura 6.18b), e que o ACF não possui um padrão bem definido, um pouco como na análise dos parâmetros não sazonais (figura 6.18a). Desta forma é estimado o parâmetro auto regressivo sazonal AR(ps)=1 e o parâmetro de média móvel sazonal MA(qs)=0.

Após a identificação de todos os parâmetros e consequente modelo ARIMA (p=1,d=1,q=0)(ps=1,ds=1,qs=0) é possível aplicá-lo e verificar a sua precisão.

Apreciação do Modelo A aplicação do modelo ARIMA identificado com os seguintes parâmetros (p=1,d=1,q=0)(ps=1,ds=1,qs=0) foi efectuada utilizando o conjunto de dados desde o primeiro mês contabilizado, ou seja Novembro de 2001 a Dezembro de 2008 para o período de apreciação do modelo e de Janeiro de 2009 até Março de 2010 para o período de previsão do modelo. A informação apresentada na tabela 6.4 visa resumir a informação necessária para a aplicação do modelo.

Tabela 6.4 Descrição do modelo ARIMA(1,1,0)(1,1,0).

		Período de Apreciação	**Período de Previsão**	**Tipo de Modelo**
Id do Modelo	Número de Defeitos por Meses	N Break Model 1	Nov 2001 - Dez 2008	Jan 2009 - Mar 2010	ARIMA(1,1,0)(1,1,0)
Outros Parâmetros				
Detectar *outliers* automaticamente - **Sim**				
Incluir Constante no Modelo - **Sim**				

Como resultado, é obtido o modelo representado na figura 6.19. Quando comparado com o modelo de ajuste (linha fina azul na figura 6.19) e os dados observados (linha vermelha na figura 6.19), parece que o modelo está ajustado o suficiente para efectuar previsões sobre períodos futuros. Analisando os valores previstos (linha espessa azul na figura 6.19) e comparando com os valores dos dados observados, podemos concluir que o modelo efectua previsões dentro de um nível de confiança aceitável, apesar de se verificar uma ligeira sobre estimação do número de defeitos na previsão, durante o período de previsão.

Para verificar a precisão do modelo, é necessário inspeccionar as suas estatísticas. Na tabela 6.5 são identificados 6 *outliers* que foram excluídos e que a precisão do modelo é razoavelmente elevada, de acordo com os valores estatísticos de R-quadrado e R-quadrado estacionário. Os valores estatísticos de erro MAPE e MaxAPE encontram-se numa margem de erro aceitável para que o modelo possa efectuar previsões de confiança. O teste de Ljung Box indica, através do valor alto da estatística Sig, que o presente modelo é apropriado e ajusta-se bem a série.

O valor estatístico de R-quadrado estacionário é uma medição que compara a parte estacionária do modelo com um modelo de médias simples. Esta medição é preferível ao R-quadrado

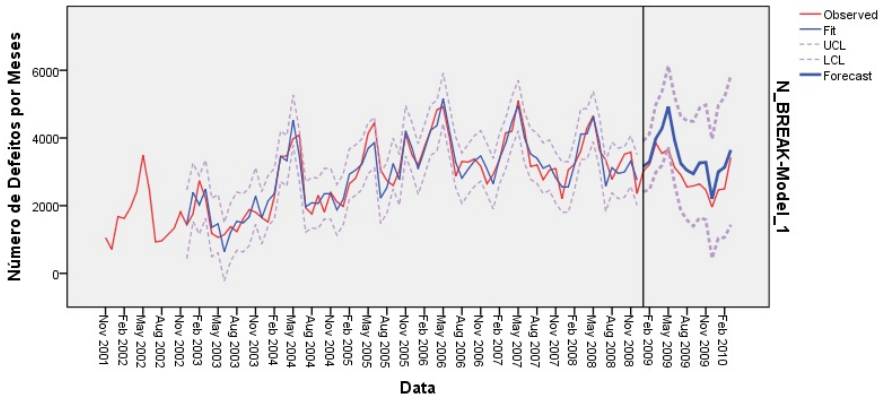


Figura 6.19 Gráfico do modelo ARIMA(1,1,0)(1,1,0).

normal quando existe uma tendência ou um padrão sazonal. O R-quadrado normal, estima a proporção da variação total da série que é explicada pelo modelo. Esta medição é mais útil quando a série é estacionária.

Tabela 6.5 Estatísticas do modelo ARIMA(1,1,0)(1,1,0).

Estatísticas do Modelo								
Modelo	Estatísticas de Ajuste do Modelo				Ljung-Box Q(18)			Nº de Outliers
	R-quadrado Estacionário	R-quadrado	MAPE	MaxAPE	Estatísticas	DF	Sig.	
110 110	,741	,871	11,558	44,353	15,443	16	,492	6

Foi aplicado o mesmo modelo, mas reduzindo agora o período de estimação para Novembro 2001 - Setembro 2008 e o período de previsão passou a ser Outubro 2008 - Março 2010. Os resultados obtidos são ligeiramente melhores em quase todas as estatísticas mas apresentam uma subtil subida na estatística MAPE, o erro absoluto médio percentual, e no teste de Ljung Box verifica-se uma ligeira diminuição, não alterando no entanto a conclusão sobre o modelo, obtido na experiência anterior, de que o modelo é, apropriado e ajusta-se bem à série.

Tabela 6.6 Estatísticas do modelo ARIMA(1,1,0)(1,1,0) para o período de estimação Nov2001 - Set2008.

Estatísticas do Modelo								
Modelo	Estatísticas de Ajuste do Modelo				Ljung-Box Q(18)			Nº de Outliers
	R-quadrado Estacionário	R-quadrado	MAPE	MaxAPE	Estatísticas	DF	Sig.	
110 110	,755	,876	11,493	44,478	16,090	16	,447	6

Resolvemos de seguida gerar modelos com outros parâmetros. Mantivemos os parâmetros de diferenciação e diferenciação sazonal, e variando os parâmetros p, q, ps e qs. Através dos

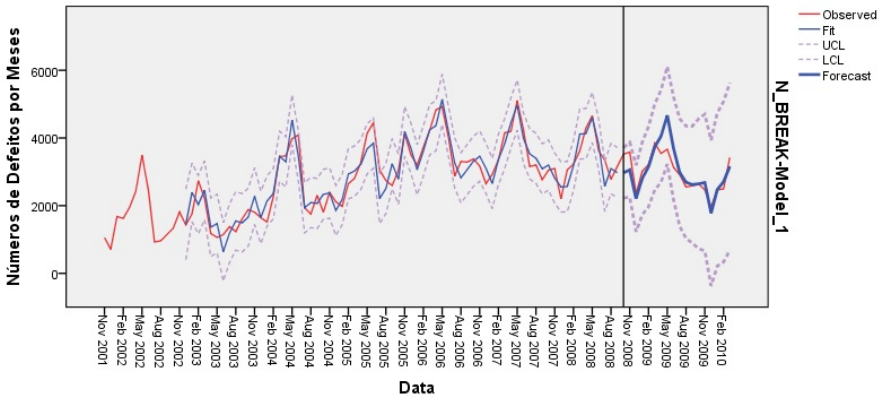


Figura 6.20 Gráfico do modelo ARIMA(1,1,0)(1,1,0) para o período de estimação (Nov2001 - Set2008).

novos modelos gerados surge o modelo ARIMA (110)(010) de estatísticas apresentadas na tabela 6.7 e representado graficamente na figura 6.21.

Tabela 6.7 Estatísticas do modelo ARIMA(1,1,0)(0,1,0).

Estatísticas do Modelo							
Modelo	Estatísticas de Ajuste do Modelo				Ljung-Box Q(18)			Nº de Outliers
	R-quadrado Estacionário	R-quadrado	MAPE	MaxAPE	Estatísticas	DF	Sig.	
110 010	,701	,851	11,663	39,673	25,812	17	,078	6

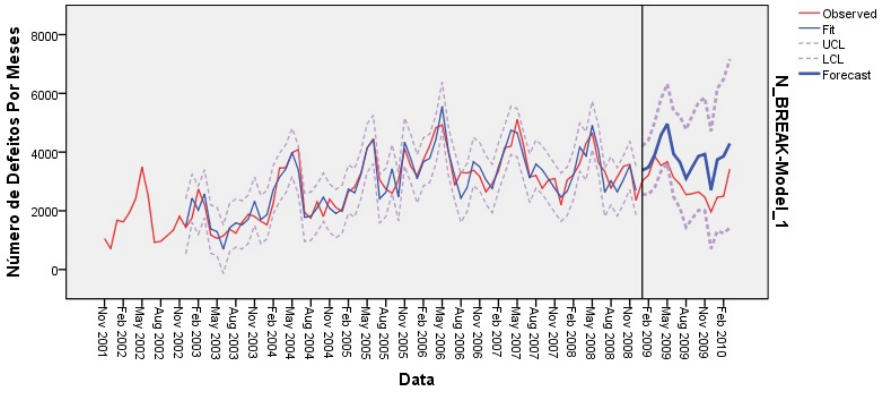


Figura 6.21 Gráfico do modelo ARIMA(1,1,0)(0,1,0).

Através dessa análise pode-se verificar que apesar das estatísticas de ajuste do modelo serem inferiores respeitante aos modelos anteriores, no entanto o máximo erro absoluto médio percentual (MaxAPE) é inferior ao verificado para os restante modelos. De salientar também

para a acentuada diminuição no teste de Ljung Box que demonstra que o este modelo não é apropriado e não se ajusta tão bem à série comparando com o primeiro modelo estudado.

Validação do modelo (H3 e H4) Finalmente foi decidido comparar o presente modelo com um modelo caminho aleatório (*Random Walk*). Neste modelo os parâmetros auto regressivos (AR) e de média móvel (MA) não são incluídos. Tem a forma de ARIMA (0,d,0)(0,ds,0). Dado que a presente série em estudo foi diferenciada uma vez não sazonalmente e uma vez sazonalmente, será utilizado o modelo ARIMA(0,1,0)(0,1,0).

Tabela 6.8 Descrição do modelo ARIMA(0,1,0)(0,1,0).

			Período de Apreciação	**Período de Previsão**	**Tipo de Modelo**
Id do Modelo	Número de Defeitos por Meses	N Break Model 1	Nov 2001 - Dez 2008	Jan 2009 - Mar 2010	ARIMA(0,1,0)(0,1,0)
Outros Parâmetros					
Detectar *outliers* automaticamente - **Sim**					
Incluir Constante no Modelo - **Sim**					

É possível concluir através das estatísticas apresentadas na tabela 6.9 que o respectivo modelo é menos robusto que os dois modelos apresentados na secção 6.5.3.2. Apesar da diferença de estatísticas de erro MAPE e MaxAPE não serem muito significativas, a diminuição na estatística do R-quadrado estacionário bem como dos valores resultantes do teste de Ljung Box, inferem com maior significância sobre a menor robustez do presente modelo respeitante aos modelos anteriores. O presente modelo apresenta ainda um valor inferior na Sig.

Tabela 6.9 Estatísticas do modelo ARIMA(0,1,0)(0,1,0).

Estatísticas do Modelo								
Modelo	Estatísticas de Ajuste do Modelo				Ljung-Box Q(18)			Nº de Outliers
	R-quadrado Estacionário	R-quadrado	MAPE	MaxAPE	Estatísticas	DF	Sig.	
010 010	,519	,760	13,781	55,552	29,063	18	,048	4

De todos os modelos avaliados, cujos os resultados são apresentados na tabela 6.10, pode-se concluir que o modelo Modelo 1, que corresponde ao ARIMA (1,1,0)(1,1,0) foi o que mostrou valores mais precisos. Recordando que o modelo Modelo 1A corresponde ao mesmo modelo ARIMA, mas em que o período de apreciação foi reduzido com o propósito de verificar eventuais alterações significativas. Foi ainda o modelo que mostrou melhores valores no teste de Ljung Box indicando com alguma segurança que o modelo escolhido é apropriado e o que mais se ajusta à série em questão.

Para além da validação através da comparação deste modelo com o modelo passeio aleatório, foram seguidamente analisados os resíduos do modelo, para reconfirmar e validar a precisão

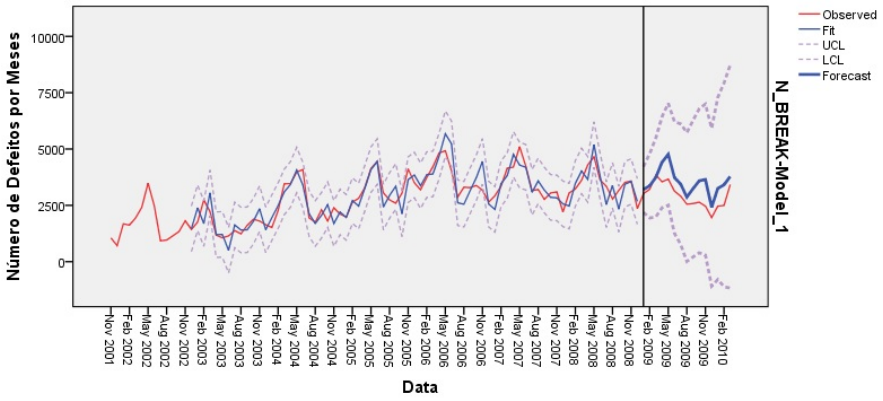


Figura 6.22 Gráfico do modelo ARIMA(0,1,0)(0,1,0).

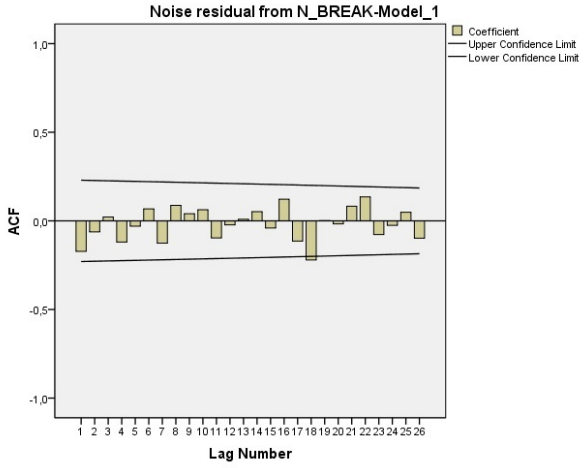
Tabela 6.10 Comparação dos modelos avaliados.

Estatísticas do Modelo								
Modelos ARIMA	Estatísticas de Ajuste do Modelo				Ljung-Box Q(18)			Nº de Outliers
	R-quadrado Estacionário	R-quadrado	MAPE	MaxAPE	Estatísticas	DF	Sig.	
Modelo 1 - (110)(110)	,741	,871	11,558	44,353	15,443	16	,492	6
Modelo 1A - (110)(110)	,755	,876	11,493	44,478	16,090	16	,447	6
Modelo 2 - (110)(010)	,701	,851	11,663	39,673	25,812	17	,078	6
Modelo 3 - (010)(010)	,519	,760	13,781	55,552	29,063	18	,048	4

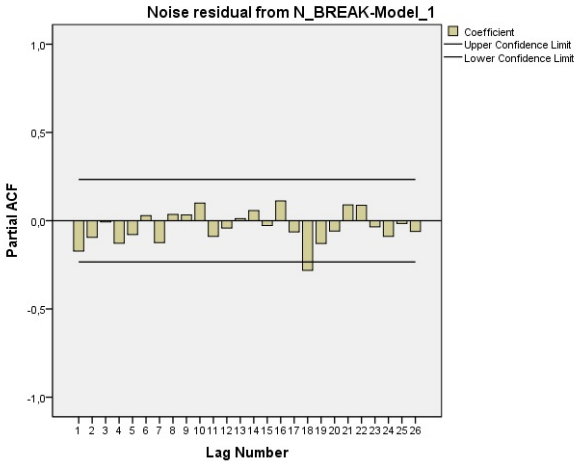
das previsões. Para concluir a validação do presente modelo é necessário verificar os resíduos da série gerados pelo modelo. O procedimento de apreciação assume que os resíduos não são auto correlacionados e que seguem uma distribuição normal. Por esta razão foram gerados diagramas necessários para confirmar a distribuição normal dos resíduos, de forma a concluir a validação do presente modelo em estudo.

A suposição de que os resíduos não estão correlacionados, é apoiada pelo facto dos valores de ACF e PACF do resíduos para cada intervalo de tempo se apresentarem dentro do intervalo de confiança e tenderem para o zero (figura 6.23). Na figura 6.24 os dois diagramas apresentados confirmam que os resíduos seguem uma distribuição normal, sendo possível verificar este facto pela forma de sino do histograma (figura 6.24a) e pela distribuição pouco dispersa dos pontos no QQ *plot* (figura 6.24b). Por fim o gráfico da dispersão dos pontos apresentado pela figura 6.25, confirma que os resíduos são observações aleatórias, devido à aleatoriedade da distribuição dos pontos, e como tal, não se correlacionam. Tendo todas estas condições satisfeitas, torna-se possível finalmente validar o presente modelo em estudo, significando que as respectivas previsões são precisas e de confiança.

Em suma, o modelo ARIMA (110)(110) passou em todos os testes necessários para que

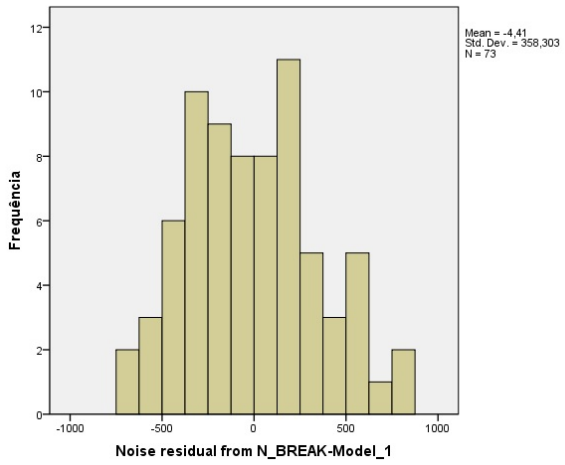


(a) ACF do ruído residual.

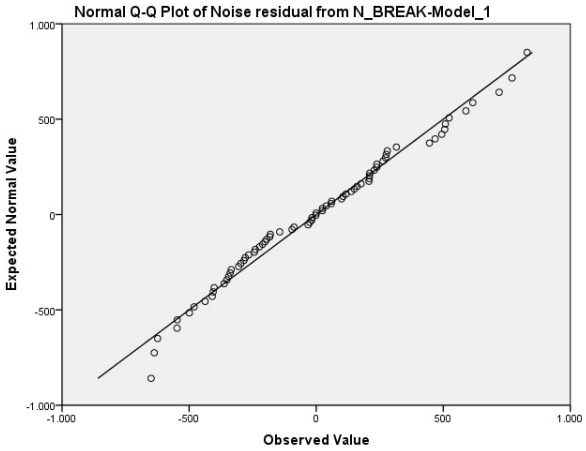


(b) PACF do ruído residual.

Figura 6.23 ACF e PACF do ruído residual.



(a) Histograma do ruído residual.



(b) QQ *plot* do ruído residual.

Figura 6.24 Histograma e QQ *plot* do ruído residual.

fosse considerado um modelo apropriado. Foi feita uma comparação com outros modelos que demonstrou que é o mais adequado em todos os aspectos e métricas respeitante aos restantes modelos. Foi também elaborada uma análise aos seus resíduos que demonstrou que o presente modelo se encontra nas condições ideais para poder afirmar que as suas previsões são válidas e de confiança.

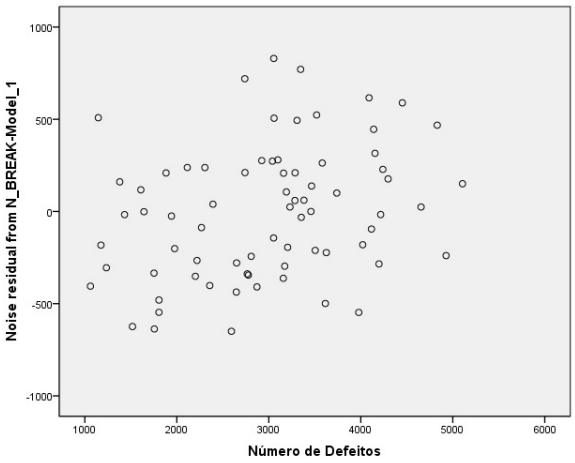


Figura 6.25 Dispersão dos pontos do ruído residual.

6.5.3.3 Análise do tempo de Resolução dos Defeitos (H5 e H6)

Para a realização desta análise considera-se que para que um defeito seja considerado resolvido, terá de conter no atributo *resolution num*, umas das possíveis identificações de resolução de defeito. Estas possíveis identificações são discriminadas na figura A.5 no anexo A. Caso nenhuma identificação se verifique, ou seja, se o atributo *resolution num* for desprovido de qualquer identificação, isso significaria que o respectivo defeito ainda não terá sido analisado. Com base nesta regra, foram extraídos todos os defeitos com uma identificação válida através de um filtro, tendo-se efectuado uma análise a evolução da distribuição da variável *duration solving*, ao longo do tempo, com base na distribuição dessa variável dentro de cada mês, em que a variável *duration solving* foi a variável dependente e *MonthYear* a variável independente. Desta forma foi possível analisar a distribuição da variável *duration solving* ao longo do tempo.

A figura 6.26 é apoiada pela tabela C que se encontra no anexo C. As barras vermelhas representam a variação da duração de resolução de defeitos de cada mês e o segmento de recta preto, corresponde a mediana da duração necessária para a resolução dos defeitos.

Após a análise do diagrama da figura 6.26 não é possível concluir a existência de um padrão sazonal ou uma tendência crescente ou decrescente ao longo do tempo, podendo-se apenas verificar que existem subidas e descidas frequentes ao longo do tempo. Este comportamento poderia indicar um padrão sazonal mas, como é possível verificar, as marcas temporais em que ocorrem aumentos e decréscimos não coincidem anualmente, pelo que não detectamos nenhum padrão sazonal.

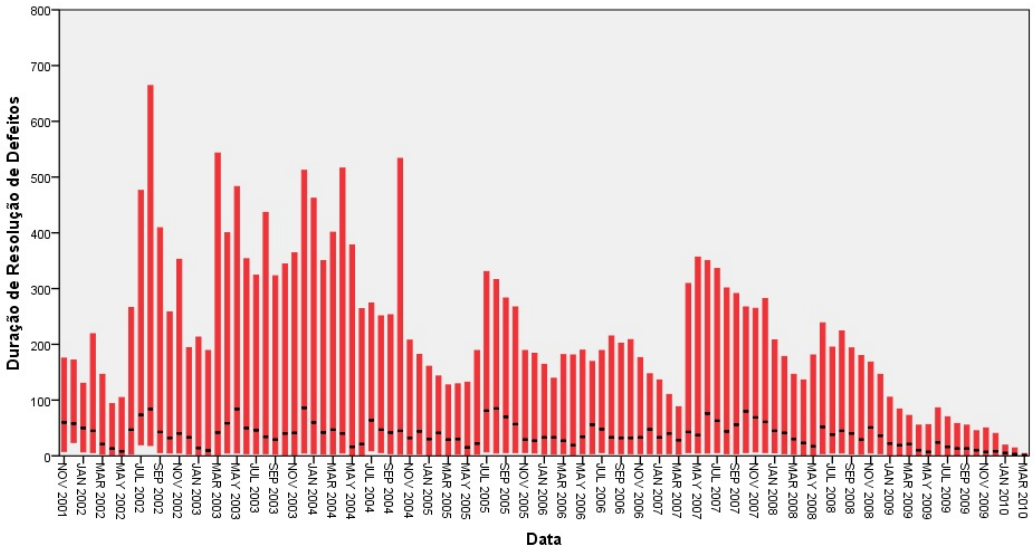


Figura 6.26 Duração da Resolução dos Defeitos ao longo do tempo.

6.5.3.4 Experiência da Janela Deslizante (H7)

Uma das questões de investigação, **QI7**, é procurar saber se a eficiência dos modelos ARIMA se mantém, numa janela temporal de dimensão fixa, de pontos de dados da história do software, quando é deslocado o ponto inicial dessa janela. Para responder a esta pergunta, procedeu-se à experiência de manter uma janela deslizante de dimensão fixa. Esta janela deslizante é composta por dois períodos, o período de estimação (PE), que é sempre composto por 74 pontos de dados correspondentes a 74 meses e o período de previsão (PP) de 12 meses. Após a realização de uma iteração, avançaram-se ambos os períodos, isto é, 1ª iteração - PE Nov2001-Dez2007, PP Jan2008-Dez2008, 2ª iteração - PE Dez2001-Jan2008, PP Fev2008-Jan2009, e assim sucessivamente até o período de previsão atingir a última marca de comparação possível que é Março de 2010.

Após a realização da totalidade desta experiência, foi gerada a tabela B.1 que se encontra no anexo B. Nesta tabela constam as estatísticas resultantes da criação do modelo para cada iteração. De seguida foram calculados as respectivas médias para comparação com o modelo da experiência anterior descrito na secção 6.5.3.2. O resumo destas estatísticas é representado na tabela 6.11.

Através da análise da tabela 6.11 é possível verificar que a média das estatísticas de ajuste da experiência com janela deslizante são menos robustas em comparação com as estatísticas

Tabela 6.11 Resumo e Comparação da Experiência Janela Deslizante.

	ARIMA 110 110			
	Janela Deslizante			Sem Janela Deslizante
	Pior	Média	Melhor	
S R-quadrado	0,324455	0,440663469	0,733731	0,741159982
R-quadrado	0,660391	0,691866402	0,858992	0,870844046
MAPE	15,50108	13,76057141	11,97149	11,55825168
MaxAPE	128,2368	52,78249692	39,56698	44,35303133
Sig	0,022685	0,149350847	0,326249	0,492477975

da experiência sem janela deslizante. No caso do R-quadrado estacionário da janela deslizante permite perceber que a média dos modelos não se ajusta nem a metade da série temporal e é menos robusto em aproximadamente 40 % em relação ao modelo sem janela deslizante. As estatísticas de erro, também são de robustez inferior para a janela deslizante. A diferença na percentagem média de erro absoluto MAPE é algo significativa, mas a diferença na percentagem de erro absoluto máximo é mais significativa, resultando numa diminuição de robustez de aproximadamente 20%. No teste de Ljung Box verificou-se que o modelo sem janela deslizante tem melhor valor do que na média dos modelos gerados na janela deslizante. Tal facto significa que o modelo sem janela deslizante reúne melhores valores em todas as estatísticas analisadas. É possível reforçar que até os valores das melhores ocorrências da janela deslizante, não são tão robustos como os valores do modelo que é usado para comparação.

6.6 Interpretação

6.6.1 Avaliação dos Resultados e Implicações

Nesta secção serão discutidos os resultados obtidos com a finalidade de responder às questões de investigação propostas nos objectivos do trabalho.

Através do trabalho efectuado foi verificado que a maioria dos defeitos é reportado nos dias de semana laborais e muito poucos ao fim de semana. Recorde-se ainda que o Eclipse é um projecto *open source* de grandes dimensões, que combina voluntários com profissionais no seu desenvolvimento, pelo que é de esperar tal distribuição dos defeitos ao longo da semana. Isto contraria a percepção errada segundo a qual o software *open source* é desenvolvido tipicamente nos tempos livres dos programadores que para ele contribuem. Deste modo, seria expectável uma maior uniformidade na distribuição dos relatórios de defeitos ao longo da semana. Por

outro lado, é necessário também considerar que muitos relatos de erros serão enviados por utilizadores "normais", sendo natural que esses concentrem mais a sua actividade durante a semana. Esta característica permite ainda comparar o presente estudo com projectos comerciais semelhantes. Foram identificados na presente amostra de estudo dois padrões sazonais que respondem a **QI1** sobre a possível existência de padrões sazonais: um padrão muito forte semanal e um padrão anual não tão claro mas que se verifica através da análise das versões do Eclipse apresentada na figura 6.7 da secção 6.5.1. De salientar que o padrão sazonal semanal é irrelevante para a presente série temporal, devido à granularidade escolhida ser meses. Sobre o padrão sazonal anual, é possível dizer que este acontece desde o início do Eclipse, não sendo possível no entanto dizer que tem acontecido sempre na mesma altura, devido a existência de três *milestones* da versão 2.1. Contudo, à excepção desta particularidade, o comportamento tem sido sempre igual, o que significa que, para além dos picos de número de defeitos ocorrerem sempre antes do lançamento de uma nova versão, esta ocorre quase sempre nos meses de Maio e Junho. Este padrão é consistente com o mês escolhido para o lançamento de novas versões do Eclipse, o mês de Junho, com excepção da versão 2.1 que foi lançada em Março de 2003. É possível também verificar que a distribuição dos defeitos é maior no primeiro semestre do ano do que no segundo. Tal facto resulta principalmente da diminuição de defeitos a seguir ao mês de Junho, que provavelmente resulta do período de férias de Verão, e também à diminuição de números de defeitos no final do ano, que por sua vez se deve provavelmente ao Natal e Ano Novo.

Uma outra particularidade que deveria ser averiguada, é a de que se a presente série temporal demonstra alguma tendência ao longo do tempo, respondendo assim a **QI2**. De facto pelo estudo e análise feita sobre esta questão, verifica-se que a presente série apresenta inicialmente um comportamento de subtis subidas e descidas, seguido de uma descida acentuada, provavelmente devido a existência de 3 *milestones* da versão 2.1 (dados disponíveis na cronologia). A partir deste ponto verifica-se um aumento contínuo com graus de intensidades maiores, que correspondem à versão 3.0 e 3.1, mantendo-se estacionária desde então. A versão 3.0, representou uma autêntica revolução no núcleo do Eclipse, devido à mudança na arquitectura do sistema de tempo de execução para o *OSGi Service Platform*. Esta mudança provocou grandes alterações na evolução do Eclipse, como se pode ver na figura 6.27, em que se pode verificar uma grande mudança entre a versão 2.1.3 e a 3.0, e em que muitas dependências externas foram removidas, e poucas ou nenhumas mantidas.

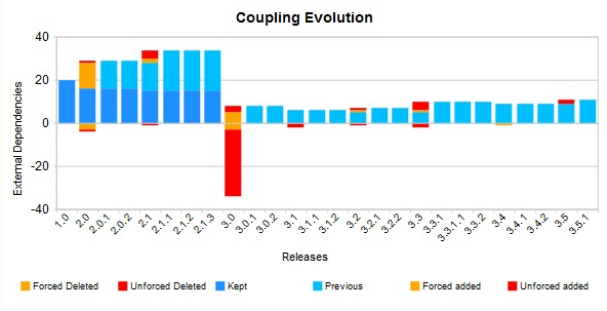


Figura 6.27 Análise da evolução de acoplamento das dependências externas. Adaptado de [81].

Esta revolução também pode ser verificada através da visualização do vídeo disponibilizado [6] por Ogawa. O vídeo foi elaborado por uma ferramenta chamada de code swarm, que foi apresentado por Ogawa et *al.*, e que mostra a história de submissões de um projecto de software [62]. Após este período agitado na história do Eclipse, a tendência da série, como já referido, tem-se mantido estacionária, provavelmente devido à consolidação desta arquitectura, sem terem existido entretanto outras revoluções de maiores dimensões.

Sobre a **QI3** é possível confirmar que os modelos ARIMA são uma aproximação válida para prever a evolução do número de defeitos, devido as boas estatísticas, tanto de ajuste do modelo como às de erro para MAPE e MaxAPE. O número médio de defeitos por mês na nossa série temporal é de 2795,40. A percentagem de erro médio na previsões do modelo foi de 11,56 %, representando um possível cálculo erróneo de aproximadamente 323 defeitos por mês, sendo no entanto uma margem de erro aceitável comparando com o possível cálculo erróneo de aproximadamente 385 defeitos do modelo caminho aleatório. Este resultado responde a **QI4**, e sim, de facto, devido à reduzida percentagem de erro do modelo, é possível ser-se confiante na utilização dos modelos ARIMA para a previsão do número de defeitos.

Respondendo às questões de investigação **QI5** e **QI6**, que procuram saber se a resolução de defeitos ao longo do tempo apresenta um padrão sazonal ou uma tendência respectivamente, foi já dito na explicação do diagrama 6.26, que não foi possível identificar um padrão sazonal ou uma tendência. Isto porque o diagrama apresenta um comportamento ao longo do tempo de várias subidas e descidas. Se tal comportamento se verificasse nas mesmas alturas dos vários anos sucessivos, teríamos um padrão sazonal, mas como este comportamento ocorre um pouco ao acaso ao longo dos anos, não nos é possível provar essa existência. De salientar apenas o pormenor de que, se forem apenas contabilizados os anos de 2007, 2008, 2009 e início de 2010,

[6] http://vidi.cs.ucdavis.edu/research

será possível verificar um padrão sazonal anual, em que o pico ocorre nos meses de Junho e Novembro de cada ano, e a seguir a cada pico ocorre uma descida gradual na duração de resolução de defeitos. Mas como neste estudo é analisada a totalidade dos dados registados no Bugzilla, excluindo os casos anormais referidos anteriormente, a descoberta deste pequeno padrão sazonal é relativamente insignificante em relação à falta de um padrão ou de uma tendência presente na cronologia do estudo realizado. Levanta, no entanto, a hipótese de que a partir de 2007 esteja a surgir um novo padrão sazonal nesta série temporal, que não será investigado no âmbito desta dissertação.

O resultado da experiência da janela deslizante indica que um modelo pode ser fiável e gerar previsões de confiança com k pontos de dados para o período de estimação e efectuar n previsões. Se forem constantemente alterados os dados históricos mais antigos, estas mudanças, acabarão por tornar o modelo pouco eficiente, e provavelmente outro modelo será então mais adequado para gerar previsões com base naqueles pontos de dados. Isto deve-se essencialmente a importância dos dados históricos para o bom enquadramento do modelo na série temporal, e esta experiência prova que para além da eficiência se perder ao longo do tempo e das sucessivas mudanças, os dados mais antigos são importantes. Esta conclusão é possível devido à queda abrupta nas estatísticas de ajuste e de erro do modelo, sem nunca conseguir voltar às estatísticas iniciais. É também possível concluir que nenhum modelo ARIMA está livre de um breve estudo inicial. Esse estudo inicial servirá para averiguar se o modelo proposto é adequado ou não para os pontos de dados fornecidos. Como resposta à questão de investigação **QI7**; a eficiência do modelo ARIMA não se mantém quando deslocado consecutivamente o ponto inicial de uma janela temporal de dimensão fixa de pontos de dados da história do software. Isto implica que é preferível que sejam mantidos os dados históricos mais antigos, em vez de serem eliminados aquando na utilização de um modelo deste tipo para a previsão de defeitos reportados.

Comparando os resultados do presente estudo com os trabalhos mencionados, conclui-se que a utilização de séries temporais para efeitos da análise da evolução de software como feito por Mens et *al.* [59] e Wermelinger et *al.* [81] é uma técnica muito útil que permite exprimir a evolução de muitos anos em apenas uma imagem. Comparando agora com o trabalho de Raja et *al.* [68], também é possível prever o número de defeitos do software escolhido. Não é permitido comparar resultados, devido ao presente trabalho ter uma componente sazonal e por isso seis parâmetros, enquanto que no trabalho de Raja, não existe a componente sazonal, ou seja, os modelos ARIMA são de apenas 3 parâmetros. Todavia, em termos de apreciação das previsões feitas pelos modelos ARIMA em ambos os trabalhos, é possível concluir que é

uma aproximação válida para a previsão do número de defeitos. Sobre o trabalho realizado por Kläs et *al.* [39], pode-se apenas concluir que é uma técnica a ser usada com a técnica apresentada nesta experiência para colmatar a falta de dados históricos, que são necessários para que os modelos ARIMA possam efectuar as melhores previsões. Por fim, comparando o presente estudo com o trabalho de Caldeira, foram obtidos resultados semelhantes na descoberta de padrões sazonais, o que faz transparecer que o projecto Eclipse cada vez mais assume o papel exemplar de como projectos *open source* podem estar ao mesmo nível que os projectos comerciais [16].

6.6.2 Ameaças à Validade

Não existe nenhuma garantia de que, de facto, todos os defeitos sejam registados no sistema de rastreio de defeitos. É possível que alguns defeitos tenham sido identificados e resolvidos através da troca de mensagens de correio electrónico, ou outro mecanismo de comunicação informal, por parte de membros da equipa de desenvolvimento, sendo que, nesse caso, não seriam adicionados ao Bugzilla e, consequentemente, não seriam contabilizados neste estudo. Esta ameaça pressupõe que no desenvolvimento de software possa existir uma troca recíproca de informação entre os membros da equipa de desenvolvimento. Num projecto *open source* de grande dimensão, com participantes distribuídos geograficamente, como o Eclipse, o peso deste tipo de comunicação informal tenderá a ser menor, o que mitiga este risco. De resto, este menor peso de comunicação informal foi já verificado num estudo realizado por Herbsleb et *al.*, em que se conclui que a comunicação informal entre membros da equipa de desenvolvimento de um projecto diminui com a distância [37].

O simples facto de ter sido escolhido um software *open source* pode ser considerado uma ameaça aos resultados obtidos, mas comparando esses mesmo resultados obtidos com um estudo feito sobre uma grande empresa de software comercial [16], a conclusão a que é possível chegar é que muitas políticas neste software *open source* seguem padrões comparáveis aos de uma grande organização de desenvolvimento de software com fins comerciais. Isto aumenta a comparabilidade do presente estudo com outros projectos comerciais, fortalecendo também assim a sua própria validade e a dos seus resultados.

6.6.3 Inferências

Nesta experiência foi possível identificar e caracterizar a distribuição do número de defeitos do Eclipse por dias registados no Bugzilla, tendo-se descoberto que esta apresenta um padrão sazonal semanal semelhante a um estudo feito aos incidentes resolvidos em projectos comerciais [16]. Este padrão distribui o número de defeitos e número de incidentes nos dias laborais e decai bruscamente no fim de semana. Este comportamento num projecto comercial é usual, mas num projecto *open source*, este padrão poderia ser esperado menos diferenciado. Este resultado veio aumentar a validade da presente experiência uma vez que permite comparar resultados de um projecto *open source* com projectos comerciais. Esta comparação com projectos de índole comercial torna estes resultados mais relevantes para a indústria. Ou seja, deixam de interessar apenas a quem se preocupa com o Eclipse, em particular, ou com software *open source*, em geral, e passam a interessar também a quem desenvolve software comercial.

6.6.4 Lições Aprendidas

Quando se realizam experiências onde é necessário recorrer ao uso de ferramentas que acabam por se tornar essenciais para o bom desenrolar da experiência, é através do seu uso e prática que pormenores e mesmo “truques” são descobertos para que a utilização dessas ferramentas se torne mais fácil para quem mais tarde venha a realizar outras experiências usando as mesmas ferramentas. Estes detalhes e "truques"tornam-se particularmente importantes por mitigarem problemas relacionados com o grande volume de dados tratados nesta experiência. Esta secção descreve alguns desses detalhes, e sugestões, para benefício de leitores que venham um dia mais tarde a recorrer a estas ferramentas nas suas actividades de experimentação.

Através da realização desta experiência algumas lições foram aprendidas sobre a ferramenta Navicat Premium. Para além do processo já explicado de inserção de defeitos numa base de dados auxiliar, para que de seguida seja extraída a informação e só depois seja finalmente inserida na base de dados final, o número de defeitos envolvidos em cada uma destas operações é importante. Como já referido anteriormente, no processo de inserção de defeitos na base de dados auxiliar foram usados blocos de 2500 defeitos por questões de dificuldades de escalabilidade das ferramentas usadas. Por exemplo, usando o dobro dos defeitos (5000), o processo de inserção aumenta para o triplo, o intervalo de tempo necessário para o processamento. O mesmo acontece na operação de extracção da informação para os ficheiros de texto, mas com subtis melhorias. Neste caso, optou-se por fazer a extracção de 2500 defeitos de cada vez, juntar 5000

defeitos num só ficheiro de texto e só depois, importá-lo na base de dados final. Neste procedimento a diferença de tempo gasto na inserção de ficheiros de texto com 2500 defeitos ou com 5000 defeitos é mínima. Estas operações são todas necessárias já que a ferramenta não permite a inserção de vários ficheiros XML para uma única tabela.

6.7 Conclusões e Trabalho Futuro

6.7.1 Sumário

Através da realização desta experiência é possível responder a todos os objectivos propostos. Verificámos que a presente série temporal de granularidade meses, apresenta um padrão sazonal anual, em que os picos ocorrem sempre no mês em que é lançada uma nova versão do Eclipse. Foi também encontrado um padrão sazonal semanal na série temporal de granularidade dias que, ao ser comparado com um estudo feito sobre projectos comerciais, obteve resultados semelhantes. A consistência neste resultado veio reforçar a ideia que alguns projectos *open source*, particularmente projectos de grande dimensão, como o Eclipse, são comparáveis com outros projectos desenvolvidos em contextos comerciais.

Foi também possível verificar a existência de uma tendência na presente série temporal. A tendência inicial é de subidas e descidas semelhantes mas a partir de 2004, é crescente até finais de 2005, e desde então tem vindo a manter-se constante. Este crescimento deve-se principalmente a revolução na arquitectura do tempo de execução do Eclipse que mudou para o *OSGi Platform Service*.

Esta experiência veio comprovar que a previsão do número de defeitos pode ser efectuada com os modelos ARIMA e que essas previsões, possuem uma margem de erro aceitável, tornando-as fiáveis e de confiança. Foi efectuada também uma análise à evolução do tempo de resolução de defeitos ao longo do tempo, em busca de padrões sazonais e tendências, mas nenhum padrão sazonal, nem nenhuma tendência foram encontrados.

Por fim foi efectuada a experiência da janela deslizante para verificar se a eficiência do modelo ARIMA se mantinha numa janela de dimensão fixa de pontos de dados históricos quando deslocávamos o início da janela, e como resultado, verificámos que nesse ambiente, a eficiência vai-se perdendo conforme os dados históricos mais antigos se vão perdendo.

6.7.2 Impacto

Esta experiência veio demonstrar que os modelos ARIMA são uma aproximação válida para a previsão de defeitos. Num contexto mais geral, veio demonstrar que pode ser usada com sucesso no controlo do processo de software, aumentando assim a sua previsibilidade, e consequentemente, a capacidade para o gerir. A utilização desta ferramenta é do interesse dos gestores de projectos, mas apenas pode ser usado com um conjunto razoável de dados históricos. Para colmatar essa ausência num projecto novo, sugerimos, que numa fase inicial se use uma técnica híbrida como em [39], substituindo-a ao fim de tempo suficiente por esta, que tem a vantagem de não estar dependente da opinião de peritos. Esta ferramenta permitirá ao gestor de projecto, a capacidade de distribuir os recursos estrategicamente para que possíveis problemas futuros sejam antecipados e assim o número de defeitos resultantes após o lançamento do produto final seja o mais reduzido possível.

De salientar também que os resultados são consistentes com os observados em software comercial, o que significa que, pelo menos para este fenómeno, o Eclipse se revelou um exemplo do mundo *open source* que permite realizar estudos cujos resultados são extrapoláveis para projectos comerciais. Esta semelhança entre este projecto *open source* e projectos comerciais vem reforçar a validade desta experiência e o impacto dos seus resultados.

6.7.3 Trabalho Futuro

Como trabalho futuro propomos efectuar esta mesma experiência para outro software, que use o Bugzilla para o registo de defeitos, verificando se existe um comportamento semelhante no que diz respeito aos picos do número de defeitos, para comprovar a verdadeira causa desses picos. Como o Bugzilla aloja os defeitos de um grande número de projectos, é aconselhado escolher um projecto que tenha um número razoável de pontos de dados da história, para que a utilização dos modelos ARIMA seja o mais eficiente possível. Por exemplo, o Mozilla seria um bom candidato à realização de tal replicação.

Outra proposta seria criar uma variável que identificasse os defeitos reportados por membros da equipa de desenvolvimento. É de assumir que os membros da equipa de desenvolvimento, são utilizadores a quem tenha sido dirigido um defeito para a sua resolução. Após a criação dessa variável, seria interessante analisar, ao longo do tempo, o contributo dos membros das equipas versus o contributo dos utilizadores normais e verificar se existe algum padrão ou tendência nessa distribuição.

Por exemplo, verificar se existiria uma maior concentração de relatos originados pela equipa de desenvolvimento nas fases que antecedem aos lançamentos das versões, contrastando com problemas reportados pelos utilizadores depois de o software ser disponibilizado. Outra avaliação interessante seria perceber qual a percentagem relativa de problemas reportados pelos utilizadores, porque isso permitiria aferir indirectamente a percepção da qualidade que estes têm do produto.

Sugerimos também a realização de uma experiência sobre um software que seja totalmente controlado pelos utilizadores. Isto para que possam introduzir tratamentos e avaliar o seu impacto nas séries temporais. Ao falar da introdução de tratamento é por exemplo uma nova técnica, ou ferramenta utilizado no processo evolutivo do software. A ideia seria ter duas equipas a desenvolver o mesmo software mas um com a nova técnica, que esta representada pelo **X** e outra equipa sem recorrer a essa nova técnica.
Por exemplo o desenho dessa experiência seria:

O X O

O O

Depois de efectuar a experiência poderíamos então, comparar o resultados obtidos da introdução da nova técnica, sabendo o impacto que teve no processo evolutivo do software.

7

Conclusões e Trabalho Futuro

7.1 Sumário

Das principais contribuições propostas na secção1.4, efectuámos uma análise da evolução do Eclipse, onde nos foi possível verificar a existência de padrões sazonais na submissão de defeitos no Bugzilla. Verificámos que no padrão sazonal anual, os picos ocorrem sempre antes do lançamento de uma nova versão, o que tudo indica ser uma altura agitada para a comunidade que desenvolve o Eclipse. Foi descoberto também um padrão sazonal semanal, que nós permitiu comparar resultados com outro trabalho feito numa grande empresa de software, reforçando a validade deste estudo e dos seus resultados.

O modelo para a previsão do número de defeitos que propomos é o modelo ARIMA 110 110. Este foi o modelo que melhores estatísticas de ajuste e de erro obteve, em comparação com os outros modelos gerados. Verificámos também que as suas previsões possuíam uma margem de erro aceitável, o que nos indica que essas previsões são fiáveis e de confiança, reforçando assim o uso desta técnica para a previsão no contexto da análise de evolução de software.

Analisámos também a variação ao longo do tempo da duração de resolução dos defeitos, na procura de padrões sazonais ou tendências, mas nenhum padrão, nem nenhuma tendência foram encontrados. Isto significa que o processo de resolução de defeitos se tem mantido relativamente estável, do ponto de vista do tempo de resolução dos problemas reportados. Testámos

ainda o nosso modelo ARIMA, num ambiente de janela deslizante, e verificámos que à medida que retirámos os dados históricos mais antigos, o modelo começou a perder a sua eficiência. Através desta experiência concluímos que os dados mais antigos são muito importantes, e que não podem ser considerados irrelevantes para o modelo por se tratarem de informação "muito antiga".

A abordagem usada nesta dissertação para a construção de modelos de previsão apresenta uma restrição importante: apenas se torna eficaz quando o volume de dados históricos o permite. Deste modo, é importante reconhecer que a abordagem não é adequada numa fase inicial de um projecto. Sugerimos por isso a utilização de uma técnica mista, que para colmatar a falta de dados históricos se socorre da experiência de peritos. Tal como relatado em [39], essa alternativa permite mitigar a ausência de dados históricos suficientes. A sua principal "desvantagem"é estar dependente da opinião de peritos, o que a torna mais propensa a variações de eficácia, consoante a experiência do perito e mais dispendiosa, pelo esforço exigido ao tal perito. Daí que, uma vez acumulados registos históricos suficientes, recomendemos a adopção da técnica baseada nas séries temporais, para a previsão. Quando se considerar que o volume de dados históricos for suficiente, a técnica mista poderá ser deixada de parte e utilizar apenas a técnica das séries temporais.

Um outro entregável deste trabalho é um pacote experimental reutilizável para trabalhos futuros que consideramos que poderão ser feitos, já que concluímos que muita coisa ainda pode ser alcançada com estes dados.

7.2 Impacto

Esta dissertação contribui com mais um caso de estudo de uso de séries temporais para a análise e previsão da evolução de software. Para além dessa contribuição, de referir também a verificação que os modelos ARIMA são uma aproximação válida para a previsão do número de defeitos reportados. Esta técnica pode ser usada por gestores de projectos para que possam efectuar previsões da evolução do projecto que estão a gerir, podendo ajudar em decisões de alocação de recursos.

Relembramos que, para aplicar esta técnica com sucesso, devemos ter acesso a dados históricos do software que estamos a desenvolver. Numa fase muito inicial, sem esses dados, sugerimos a aplicação de um modelo híbrido como o estudado por Kläs et *al.* [39], e que para colmatar a falta ou a reduzida quantidade de dados históricos, recorrem à experiência de peritos. Numa

fase posterior, quando os dados já forem suficientes, do género 2, 3 ou mais anos, aconselhamos então, a utilizar apenas a técnica dos modelos ARIMA para a previsão.

7.3 Trabalho Futuro

Como trabalho futuro propomos a realização do mapeamento entre os dados proveniente dos repositórios de dados (CVS) com os dados dos repositórios de defeitos (Bugzilla), de um modo semelhante ao trabalho realizado por Zimmermann et *al.* [84], mas para todas as versões do Eclipse e com mais informação sobre os defeitos do que a contagem dos defeitos antes e após o lançamento. Para que a informação contida na série temporal seja mais pormenorizada, esse aumento de informação pode fazer com que as previsões se tornem ainda mais precisas do que são actualmente, demonstrando também possivelmente novas tendências e padrões. Para além desse pormenor, este mapeamento permitiria ter uma localização mais específica das partes mais problemáticas do Eclipse, já que a informação dos defeitos indica-nos apenas a que componente aquele defeito se refere, e não propriamente a função ou parte do código responsáveis pelo defeito. Note-se que a granularidade dos componentes usada para o relato de problemas não permite uma localização dos defeitos com uma precisão suficiente para que seja feita uma análise detalhada. Tal mapeamento abriria o caminho para a análise sobre técnicas de resolução de defeitos, tais como, se são eficazes ou não, se existe alguma relação entre o número de defeitos e as técnicas usadas na sua prevenção, se alguma técnica de resolução específica apresenta tendências para resolver localmente defeitos, introduzindo problemas por efeitos colaterais noutros componentes.

Outra sugestão para trabalho futuro é efectuar uma replicação deste estudo, usando outro objecto de estudo como, por exemplo, o Mozilla. No projecto Mozilla também se registam os defeitos no Bugzilla, e sua dimensão e longevidade também o tornam adequado para um estudo deste tipo. Em contraponto, o Mozilla é construído em cima de tecnologias diferentes das usadas no Eclipse, o que é um factor de diferenciação interessante. Caso se verificasse comportamento semelhante aos nossos resultados que foram comparados com projectos comerciais, isso seria uma forma de validação cruzada dos resultados em ambas as experiências, que permitiria aprofundar o conhecimento adquirido sobre padrões de evolução de software *open source*.

Sugerimos também a continuação de estudos como o nosso para ajudar a disseminar práticas de previsão da evolução de software pelos gestores de projectos. Isto para que através

dessas previsões possam alocar da melhor maneira possível os recursos essenciais para o desenvolvimento do software, tais como, pessoas, tempo e dinheiro. Este tipo de abordagens deverá, a prazo, trazer maior previsibilidade ao processo de desenvolvimento e manutenção do software, mitigando assim uma das dificuldades históricas da indústria de software: a capacidade de estimar com precisão os recursos usados na construção e manutenção de sistemas.

Anexos

Tabelas de Apoio

As tabelas seguintes pretendem apoiar as análises e ilações efectuadas sobre a base de dados completa. A informação apresentada foi retirada directamente dos *logs* gerados pelo SPSS, onde poderemos ver a frequência em número de defeitos existentes de um certo atributo, a sua percentagem, percentagem válida e acumulada na base de dados

Tabela A.1 Frequência da distribuição dos defeitos através do atributo *classification num*

Classification Numeric

		Frequência	%	% Válida	% Acumulada
Válido	Eclipse Foundation	6488	2,3	2,3	2,3
	Eclipse	139013	48,4	48,4	50,7
	BIRT	17833	6,2	6,2	56,9
	Tools	35534	12,4	12,4	69,3
	Technology	16364	5,7	5,7	75,0
	TPTP	10652	3,7	3,7	78,7
	DSDP	4885	1,7	1,7	80,4
	WebTools	22123	7,7	7,7	88,1
	DataTools	2312	0,8	0,8	88,9
	STP	703	0,2	0,2	89,1
	Modeling	16844	5,9	5,9	95
	RT	14307	5,0	5,0	100,0
	SOA	104	0,0	0,0	100,0
	Total	287162	100,0	100,0	

A tabela A.1 pretende apoiar a figura A.1 que mostra a distribuição dos defeitos na base de dados pelo atributo *classification num*. Podemos verificar que quase metade dos defeitos tem a classificação Eclipse, e a outra metade está distribuída por outros componentes utilizados conjuntamente com o Eclipse. Podemos verificar os atributos que contém a palavra *Tools* (ferramenta), que acumuladas resultam em 20.9% que é quase metade do número de defeitos

do Eclipse. Estes valores tem vindo a subir devido a arquitectura de *plug-ins* do Eclipse fazendo com que o domínio inicial do Eclipse no número de defeitos no Bugzilla tenha vindo a desaparecer.

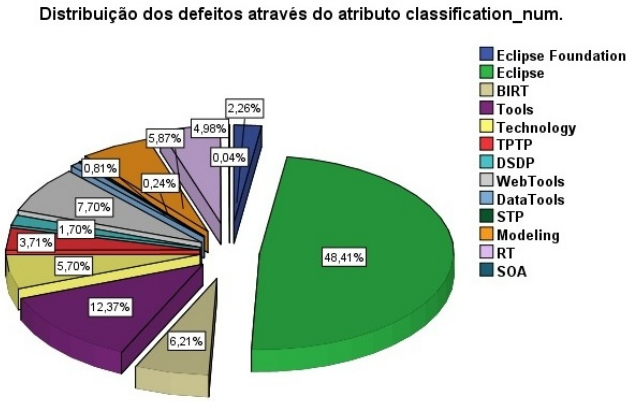


Figura A.1 Gráfico da distribuição dos defeitos através do atributo *classification num*.

		Frequência	%	% Válida	% Acumulada
Válido	blocker	4449	1,5	1,5	1,5
	critical	9231	3,2	3,2	4,8
	major	26670	9,3	9,3	14,1
	normal	189494	66,0	66,0	80,0
	minor	11599	4,0	4,0	84,1
	trivial	3756	1,3	1,3	85,4
	enhancement	41963	14,6	14,6	100,0
	Total	287162	100,0	100,0	

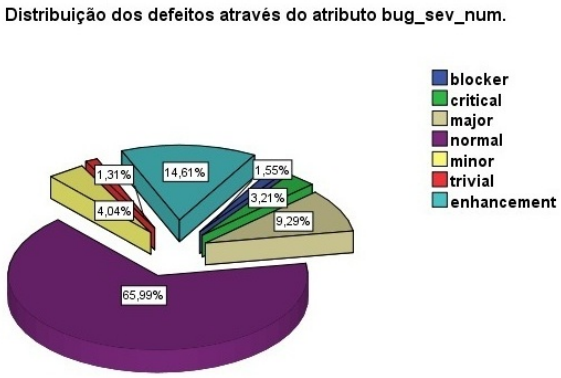


Figura A.2 Tabela e Gráfico da distribuição dos defeitos através do atributo *bug sev num*.

A figura A.2 contém a tabela e gráfico que mostra a distribuição dos defeitos na base de dados pelo atributo *bug sev num*. Este atributo mostra a severidade atribuída a cada defeito. Podemos verificar que dois terços dos defeitos possuem severidade normal, seguindo-se dos defeitos de severidade reforço (*enhancement*) e severidade principal (*major*). De salientar para o número reduzido de defeitos de severidade crítica (*critical*)e bloqueantes (*blocker*).

		Frequência	%	% Válida	% Acumulada
Valid	NEW	36190	12,6	12,6	12,6
	ASSIGNED	7831	2,7	2,7	15,3
	REOPENED	918	0,3	0,3	15,6
	RESOLVED	156695	54,6	54,6	70,2
	VERIFIED	33619	11,7	11,7	81,9
	CLOSED	51909	18,1	18,1	100,0
	Total	287162	100,0	100,0	

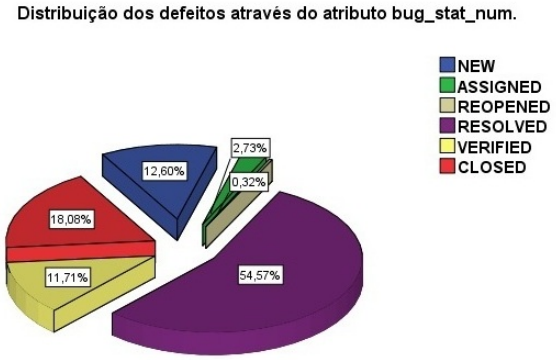


Figura A.3 Tabela e Gráfico da distribuição dos defeitos através do atributo *bug stat num*.

A figura A.3 contém a tabela e gráfico que mostra a distribuição dos defeitos na base de dados pelo atributo *bug stat num*. Este atributo mostra o estado em que se encontra cada defeito. Podemos verificar que um pouco mais de metade dos defeitos, encontram-se resolvidos. A outra metade é dividida pelos defeitos que possuem o estado novo, fechado e verificado.

		Frequência	%	% Válida	% Acumulada
Válido	P1	11169	3,9	3,9	3,9
	P2	16372	5,7	5,7	9,6
	P3	250034	87,1	87,1	96,7
	P4	5448	1,9	1,9	98,6
	P5	4139	1,4	1,4	100,0
	Total	287162	100,0	100,0	

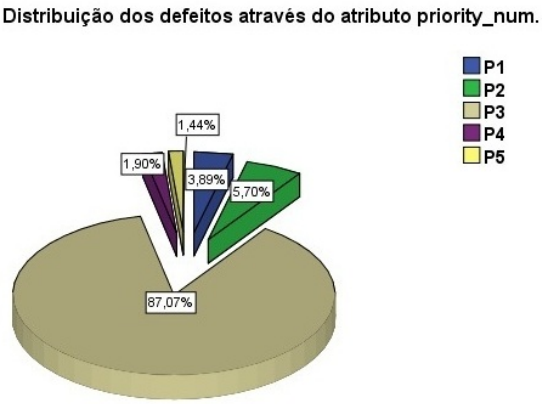


Figura A.4 Tabela e Gráfico da distribuição dos defeitos através do atributo *priority num*.

A figura A.4 contém a tabela e gráfico que mostra a distribuição dos defeitos na base de dados pelo atributo *priority num*. Este atributo caracteriza a prioridade de cada defeito. Podemos verificar que a maioria dos defeitos tem a prioridade P3, 87.1 %. Isto leva-nos a crer que os membros activos na comunicação de defeitos, acham esta classificação ambígua dando por isso o valor intermédio. Este comportamento faz com que esta variável se torne pouco útil para o nosso estudo.

		Frequência	%	% Válida	% Acumulada
Válido	DUPLICATE	34533	12,0	14,3	14,3
	FIXED	151905	52,9	62,7	77,0
	INVALID	15785	5,5	6,5	83,5
	LATER	1811	0,6	0,7	84,2
	NOT ECLIPSE	1249	0,4	0,5	84,7
	REMIND	731	0,3	0,3	85,1
	WONTFIX	18400	6,4	7,6	92,6
	WORKSFORME	17809	6,2	7,4	100,0
	Total	242223	84,4	100,0	
Em Falta	Blank	44919	15,6		
	—	20	0,0		
	Total	44939	15,6		
Total		287162	100,0		

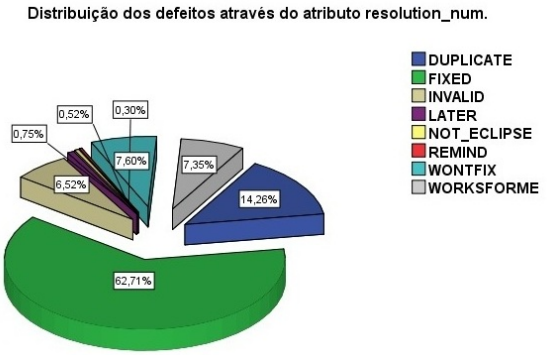


Figura A.5 Tabela e Gráfico da distribuição dos defeitos através do atributo *resolution num*.

A figura A.5 contém a tabela e gráfico que mostra a distribuição dos defeitos na base de dados pelo atributo *resolution num*. Este atributo caracteriza a resolução de cada defeito. Podemos verificar que grande parte dos defeitos são arranjados. De salientar também o número um pouco elevado dos defeitos duplicados 14.2%, isto pode traduzir-se na dificuldade de um membro saber se um defeito já foi reportado ou não, e sem ter essa informação, reporta na mesma.

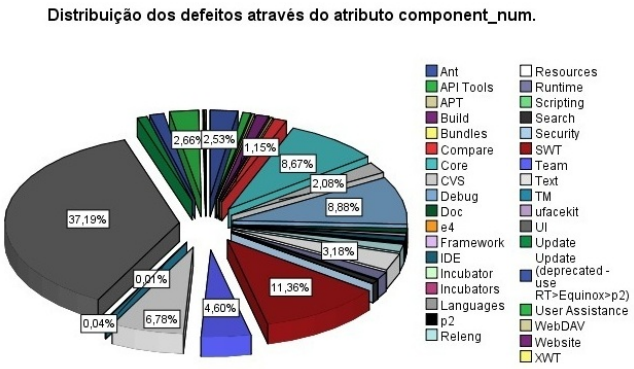


Figura A.6 Gráfico da distribuição dos defeitos através do atributo *component num*.

A tabela A.2 pretende apoiar a figura A.6 que mostra a distribuição dos defeitos na base de dados pelo atributo *component num*. Este atributo caracteriza o componente pertencente ao eclipse. Para a criação desta tabela e consequente figura, filtramos apenas os defeitos que continham o *classification num*=2, que corresponde ao Eclipse. Isto porque queríamos ver, qual o componente com maior número de defeitos. Podemos verificar que o componente com maior incidência de defeitos é o UI com 37.2 % seguindo-se do SWT com 11.4 % e Core e Debug que

rondam os 9 %. Informou-nos também que o número de defeitos do Eclipse presentes na base de dados dividem-se em 30 componentes diferentes.

Tabela A.2 Frequência da distribuição dos defeitos através do atributo *component num*.

		Frequência	%	% Válida	% Acumulada
Válido	Ant	3512	1,2	2,5	2,5
	API Tools	1030	,4	,7	3,3
	APT	327	,1	,2	3,5
	Build	1618	,6	1,2	4,7
	Bundles	313	,1	,2	4,9
	Compare	1600	,6	1,2	6,0
	Core	12056	4,2	8,7	14,7
	CVS	2890	1,0	2,1	16,8
	Debug	12344	4,3	8,9	25,7
	Doc	1122	,4	,8	26,5
	e4	41	,0	,0	26,5
	Framework	656	,2	,5	27,0
	IDE	1148	,4	,8	27,8
	Incubator	169	,1	,1	27,9
	Incubators	302	,1	,2	28,1
	Languages	21	,0	,0	28,2
	p2	15	,0	,0	28,2
	Releng	1934	,7	1,4	29,6
	Resources	4422	1,5	3,2	32,7
	Runtime	1984	,7	1,4	34,2
	Scripting	71	,0	,1	34,2
	Search	975	,3	,7	34,9
	Security	5	,0	,0	34,9
	SWT	15795	5,5	11,4	46,3
	Team	6396	2,2	4,6	50,9
	Text	9423	3,3	6,8	57,7
	TM	8	,0	,0	57,7
	ufacekit	52	,0	,0	57,7
	UI	51701	18,0	37,2	94,9
	Update	1261	,4	,9	95,8
	Update deprecated	1816	,6	1,3	97,1
	User Assistance	3696	1,3	2,7	99,8
	WebDAV	88	,0	,1	99,8
	Website	46	,0	,0	99,9
	XWT	176	,1	,1	100,0
	Total	139013	48,4	100,0	
Em Falta	System	148149	51,6		
Total		287162	100,0		

As três tabelas apresentadas de seguida, pretendem apoiar as figuras 6.9a, 6.9b e 6.10 apresentadas na secção 6.5.3.1. Estas tabelas possuem informação mais detalhada que por vezes as figuras não fazem transparecer.

Tabela A.3 Frequência do número de defeitos acumulados por semanas.

		Frequência	%	% Válida	% Acumulada
Válido	1	3107	1,1	1,1	1,1
	2	4866	1,7	1,7	2,8
	3	5245	1,9	1,9	4,7
	4	5847	2,1	2,1	6,7
	5	5695	2,0	2,0	8,8
	6	6204	2,2	2,2	10,9
	7	6349	2,2	2,2	13,2
	8	6508	2,3	2,3	15,5
	9	6273	2,2	2,2	17,7
	10	6777	2,4	2,4	20,1
	11	6859	2,4	2,4	22,5
	12	6409	2,3	2,3	24,8
	13	7132	2,5	2,5	27,3
	14	6789	2,4	2,4	29,7
	15	6014	2,1	2,1	31,8
	16	6403	2,3	2,3	34,1
	17	6630	2,3	2,3	36,4
	18	5806	2,1	2,1	38,5
	19	7042	2,5	2,5	41,0
	20	7169	2,5	2,5	43,5
	21	7752	2,7	2,7	46,3
	22	7530	2,7	2,7	48,9
	23	6886	2,4	2,4	51,3
	24	6959	2,5	2,5	53,8
	25	5715	2,0	2,0	55,8
	26	5010	1,8	1,8	57,6
	27	4217	1,5	1,5	59,1
	28	4638	1,6	1,6	60,7
	29	4695	1,7	1,7	62,4
	30	4318	1,5	1,5	63,9
	31	3941	1,4	1,4	65,3
	32	4163	1,5	1,5	66,8
	33	4040	1,4	1,4	68,2
	34	4454	1,6	1,6	69,8
	35	4178	1,5	1,5	71,3
	36	4139	1,5	1,5	72,7
	37	4376	1,5	1,5	74,3
	38	5082	1,8	1,8	76,1
	39	4765	1,7	1,7	77,7
	40	4435	1,6	1,6	79,3
	41	4519	1,6	1,6	80,9
	42	4782	1,7	1,7	82,6
	43	4646	1,6	1,6	84,2
	44	5726	2,0	2,0	86,3
	45	5484	1,9	1,9	88,2
	46	5651	2,0	2,0	90,2
	47	5358	1,9	1,9	92,1
	48	4931	1,7	1,7	93,8
	49	5235	1,9	1,9	95,7
	50	6075	2,1	2,1	97,8
	51	4366	1,5	1,5	99,4
	52	1555	,5	,5	99,9
	53	206	,1	,1	100,0
	Total	282951	100,0	100,0	

Tabela A.4 Frequência do número de defeitos acumulados por meses.

		Frequência	%	% Válida	% Acumulada
Válido	1	21589	7,6	7,6	7,6
	2	25312	8,9	8,9	16,6
	3	29696	10,5	10,5	27,1
	4	27831	9,8	9,8	36,9
	5	31024	11,0	11,0	47,9
	6	27189	9,6	9,6	57,5
	7	19588	6,9	6,9	64,4
	8	18531	6,5	6,5	71,0
	9	19456	6,9	6,9	77,8
	10	20661	7,3	7,3	85,1
	11	23505	8,3	8,3	93,4
	12	18569	6,6	6,6	100,0
	Total	282951	100,0	100,0	

Tabela A.5 Frequência do número de defeitos acumulados por dias de semana.

Dia de Semana

		Frequência	%	% Válida	% Acumulada
Válido	1	9940	3,5	3,5	3,5
	2	48092	17,0	17,0	20,5
	3	56778	20,1	20,1	40,6
	4	59084	20,9	20,9	61,5
	5	54701	19,3	19,3	80,8
	6	45543	16,1	16,1	96,9
	7	8813	3,1	3,1	100,0
	Total	282951	100,0	100,0	

Experiência Janela Deslizante

A tabela apresentada de seguida, enquadra-se na experiência de janela deslizante apresentada na secção 6.5.3.4, efectuada sobre o modelo *ARIMA 110 110*, com a finalidade de apoiar as ilações afirmadas nessa mesma secção.

Tabela B.1 Experiência Janela Deslizante, Modelo ARIMA 110 110.

							ARIMA 110 110									
	Obs	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
Jan-08	3057	2265															
Fev-08	3230	2698	3280														
Mar-08	3615	3331	4001	3959													
Abr-08	4297	3600	4248	4224	3950												
Mai-08	4655	4117	4809	4777	4448	4604											
Jun-08	3625	3204	3910	3885	3541	3787	3798										
Jul-08	3346	2063	2785	2767	2401	2625	2696	2546									
Ago-08	2778	2255	2979	2975	2599	2865	2924	2779	3571								
Set-08	3160	1975	2700	2710	2323	2614	2678	2536	3316	2359							
Out-08	3520	2147	2892	2906	2496	2781	2892	2775	3544	2629	3378						
Nov-08	3581	2051	2818	2835	2400	2672	2838	2748	3507	2663	3431	3561					
Dez-08	2358	1291	2061	2092	1646	1939	2114	2029	2776	1752	2535	2665	2682				
Jan-09	3022		2660	2688	2211	2471	2727	2681	3417	2593	3390	3520	3537	3239			
Fev-09	3214			2965	2474	2748	3023	2986	3710	2752	3564	3693	3710	3413	3203		
Mar-09	3849				3022	3485	3602	3567	4280	3124	3950	4078	4094	3801	3584	3602	
Abr-09	3544					3745	4015	4021	4724	3792	4633	4759	4776	4486	4262	4288	4527
Mai-09	3668						4625	4629	5320	4136	4991	5117	5134	4846	4617	4650	4897
Jun-09	3129							3694	4373	3092	3962	4087	4103	3819	3583	3624	3878
Jul-09	2910								3791	2799	3684	3808	3824	3543	3301	3348	3610
Ago-09	2551									2218	3117	3240	3255	2978	2729	2784	3053
Set-09	2585										3499	3622	3637	3363	3108	3170	3446
Out-09	2641											3982	3997	3725	3465	3534	3817
Nov-09	2466												4057	3789	3522	3599	3889
Dez-09	1959													2569	2296	2380	2677
Jan-10	2468														2956	3048	3352
Fev-10	2495															3243	3555
Mar-10	3426																4201
	MAPE	14,006	14,35204	13,90935	13,49307	10,93161	16,41622	15,22418	13,87744	13,74679	13,94762	13,80643	13,74406	13,51311	13,40438	12,9223	12,87453	**Média**
	MaxAPE	51,385	48,6906	49,77606	52,20452	50,9841	112,1229	93,2194	49,11539	41,44678	42,01771	42,03106	42,02413	42,28235	42,00909	42,37095	42,8399	13,76057
S	**R-quadrado**	0,568	0,5538	0,557278	0,573732	0,785116	0,375133	0,378461	0,394673	0,362497	0,355689	0,355648	0,355449	0,357676	0,356988	0,36048	0,359997	52,7825
	R-quadrado	0,781	0,763705	0,763243	0,78018	0,892264	0,703795	0,701976	0,695691	0,696081	0,660515	0,643311	0,630069	0,623404	0,599535	0,572191	0,562902	0,440663
	Sig	0,116	0,128093	0,09505	0,03787	0,342151	0,478161	0,616165	0,512205	0,011153	0,008396	0,008181	0,008574	0,006338	0,007261	0,006957	0,007058	0,691866
																		0,149351

Análise da Duração da Resolução dos Defeitos

A tabela apresentada neste capítulo visa apoiar o estudo realizado na secção 6.5.3.3 e resulta da frequência estatística efectuada sobre a variável *duration solving* com a finalidade de identificar padrões e tendências na sua distribuição ao longo do tempo.

		Mean	Lower Bound	Upper Bound	5% Trimmed Mean	Median	Variance	Std. Deviation	Min	Max	Range	Interquartile Range	Skewness	Kurtosis
Nov-01	Statistic	301,62	262,62	340,62	185,19	60	413933,433	643,377	0	2992	2992	167	2,931	7,984
	Std. Error	19,874											0,076	0,151
Dez-01	Statistic	332,07	280,27	383,86	212,33	58	489181,395	699,415	0	2871	2871	148	2,734	6,419
	Std. Error	26,379											0,092	0,184
Jan-02	Statistic	321,78	288,2	355,36	204,51	50	489835,649	699,883	0	2804	2804	124	2,675	6,018
	Std. Error	17,121											0,06	0,12
Fev-02	Statistic	429,23	388,46	470,01	324,31	45	697057,168	834,899	0	2929	2929	214	2,008	2,431
	Std. Error	20,788											0,061	0,122
Mar-02	Statistic	328,43	296,13	360,74	213,7	21	526133,433	725,351	0	2854	2854	143	2,497	4,958
	Std. Error	16,472											0,056	0,111
Abr-02	Statistic	271,38	245,77	296,99	155,07	13	409886,654	640,224	0	2802	2802	92	2,831	7,179
	Std. Error	13,06											0,05	0,1
Mai-02	Statistic	258,12	236,84	279,4	141,25	8	407928,93	638,693	0	2766	2766	103	2,892	7,331
	Std. Error	10,853											0,042	0,083
Jun-02	Statistic	375,66	347,48	403,84	270,82	47	505927,437	711,286	0	2647	2647	263	2,22	3,828
	Std. Error	14,37											0,049	0,099
Jul-02	Statistic	411,2	367,1	455,3	314,76	73,5	455462,758	674,88	0	2700	2700	456	2,009	3,217
	Std. Error	22,471											0,081	0,163
Ago-02	Statistic	471,18	424,68	517,69	381,06	84	529618,883	727,749	0	2603	2603	648	1,723	1,862
	Std. Error	23,699											0,08	0,159
Set-02	Statistic	432,62	388,52	476,71	338,84	43	563055,249	750,37	0	2674	2674	408	1,852	2,142
	Std. Error	22,472											0,073	0,146
Out-02	Statistic	376,28	338,21	414,35	278,62	32	492930,898	702,09	0	2557	2557	255	2,096	3,195
	Std. Error	19,405											0,068	0,135
Nov-02	Statistic	377,61	345,47	409,75	281,88	40	476658,12	690,404	0	2622	2622	348	2,063	3,084
	Std. Error	16,387											0,058	0,116
Dez-02	Statistic	320,75	288,1	353,4	222,51	33	389585,535	624,168	0	2612	2612	195	2,307	4,477
	Std. Error	16,646											0,065	0,13
Jan-03	Statistic	356,23	322,57	389,89	260,95	14	503614,139	709,658	0	2613	2613	213	2,083	3,004
	Std. Error	17,161											0,059	0,118
Fev-03	Statistic	312,53	287,64	337,42	214,6	9,5	427703,339	653,99	0	2455	2455	187	2,313	4,152
	Std. Error	12,695											0,048	0,095
Mar-03	Statistic	433,86	403,24	464,47	350,43	42	523743,249	723,701	0	2539	2539	545	1,78	1,865
	Std. Error	15,611											0,053	0,106
Abr-03	Statistic	383,66	345,52	421,8	296,75	58,5	426935,872	653,403	0	2465	2465	395	2,005	2,84
	Std. Error	19,438											0,073	0,145
Mai-03	Statistic	450,45	406,38	494,52	372,23	84	504275,829	710,124	0	2450	2450	479	1,719	1,605
	Std. Error	22,456											0,077	0,155
Jun-03	Statistic	387,6	348,05	427,15	304,23	50	447031,005	668,604	0	2431	2431	350	1,884	2,196
	Std. Error	20,159											0,074	0,147
Jul-03	Statistic	359,29	324,93	393,66	274,56	46	405378,546	636,693	0	2393	2393	321	2,048	2,93
	Std. Error	17,518											0,067	0,135
Ago-03	Statistic	332,01	299,44	364,59	248,73	34	326347,072	571,268	0	2234	2234	434	2,092	3,562
	Std. Error	16,602											0,071	0,142
Set-03	Statistic	301,1	274,38	327,83	217,57	29	287389,603	536,087	0	2360	2360	321	2,256	4,502
	Std. Error	13,625											0,062	0,124
Out-03	Statistic	348,17	319,72	376,62	267,99	40	376954,721	613,966	0	2315	2315	342	2,032	2,83
	Std. Error	14,504											0,058	0,116
Nov-03	Statistic	357,01	328,02	386	279,48	41	377987,185	614,807	0	2215	2215	362	1,975	2,625

Continua na página seguinte

Tabela C.1 Estatísticas da Duração da Resolução dos Defeitos. – continuação da página anterior

		Mean	Lower Bound	Upper Bound	5% Trimmed Mean	Median	Variance	Std. Deviation	Min	Max	Range	Interquartile Range	Skewness	Kurtosis
	Std. Error	14,781											0,059	0,118
Dez-03	Statistic	381,92	351,31	412,54	308,67	86	370744,945	608,888	0	2124	2124	507	1,841	2,184
	Std. Error	15,607											0,063	0,125
Jan-04	Statistic	369,78	337,87	401,69	297,26	60	377071,581	614,062	0	2232	2232	459	1,835	2,037
	Std. Error	16,267											0,065	0,13
Fev-04	Statistic	333,25	308,76	357,74	257,73	42	340436,69	583,47	0	2156	2156	347	2,004	2,75
	Std. Error	12,488											0,052	0,105
Mar-04	Statistic	372,93	351,46	394,39	303,43	47	399131,583	631,769	0	2146	2146	398	1,755	1,576
	Std. Error	10,948											0,042	0,085
Abr-04	Statistic	420	397,04	442,96	357,56	40	453479,594	673,409	0	2151	2151	511	1,508	0,63
	Std. Error	11,712											0,043	0,085
Mai-04	Statistic	350,29	330,7	369,88	281,66	16	378166,714	614,953	0	2108	2108	377	1,758	1,591
	Std. Error	9,993											0,04	0,08
Jun-04	Statistic	194,87	184,41	205,33	148,06	21	107001,619	327,111	0	1916	1916	262	2,12	4,224
	Std. Error	5,337											0,04	0,08
Jul-04	Statistic	213,3	198,05	228,55	170,47	64	105785,844	325,247	0	1367	1367	266	1,958	2,94
	Std. Error	7,777											0,059	0,117
Ago-04	Statistic	210,93	194,31	227,54	167,12	47	111006,374	333,176	0	1310	1310	245	1,919	2,661
	Std. Error	8,468											0,062	0,124
Set-04	Statistic	238,56	220,81	256,32	176,93	42	169360,366	411,534	0	1876	1876	249	2,284	4,698
	Std. Error	9,054											0,054	0,108
Out-04	Statistic	387,21	358,21	416,21	330,73	45	363836,655	603,189	0	2001	2001	531	1,467	0,578
	Std. Error	14,787											0,06	0,12
Nov-04	Statistic	183,96	170,62	197,31	137,89	32	99687,604	315,733	0	1763	1763	205	2,379	5,701
	Std. Error	6,806											0,053	0,105
Dez-04	Statistic	169,34	156,74	181,94	129,06	44	78972,595	281,021	0	1213	1213	179	2,164	3,85
	Std. Error	6,423											0,056	0,112
Jan-05	Statistic	167,07	154,01	180,14	128,74	30	78484,281	280,15	0	1193	1193	159	2,018	3,083
	Std. Error	6,663											0,058	0,116
Fev-05	Statistic	162,66	151,81	173,52	125,54	41	72608,297	269,459	0	1163	1163	141	2,031	3,155
	Std. Error	5,535											0,05	0,101
Mar-05	Statistic	155,69	145,14	166,24	118,38	29	73413,36	270,949	0	1146	1146	125	2,052	3,122
	Std. Error	5,378											0,049	0,097
Abr-05	Statistic	148,84	139,68	158	115,03	30	63331,583	251,658	0	1099	1099	126	1,98	2,907
	Std. Error	4,672											0,045	0,091
Mai-05	Statistic	139,07	131	147,13	104,09	15	63038,855	251,075	0	1071	1071	130	2,054	3,187
	Std. Error	4,113											0,04	0,08
Jun-05	Statistic	157,23	149,24	165,22	123,11	22	65153,087	255,251	0	1038	1038	186	1,884	2,578
	Std. Error	4,075											0,039	0,078
Jul-05	Statistic	196,62	187,08	206,16	169,35	81	62437,203	249,874	0	1006	1006	324	1,395	1,083
	Std. Error	4,863											0,048	0,095
Ago-05	Statistic	200,06	189,78	210,34	173,02	85	62731,806	250,463	0	983	983	311	1,355	0,936
	Std. Error	5,241											0,051	0,102
Set-05	Statistic	185,26	174,91	195,61	157,98	70	60653,161	246,279	0	944	944	277	1,457	1,059
	Std. Error	5,277											0,052	0,105
Out-05	Statistic	163,57	155,34	171,8	138,57	57	46291,814	215,155	0	917	917	261	1,515	1,538
	Std. Error	4,199											0,048	0,096
Nov-05	Statistic	138,96	132,1	145,81	111,92	29	44482,418	210,909	0	893	893	184	1,786	2,27

Continua na página seguinte

Tabela C.1 Estatísticas da Duração da Resolução dos Defeitos. – continuação da página anterior

		Mean	Lower Bound	Upper Bound	5% Trimmed Mean	Median	Variance	Std. Deviation	Min	Max	Range	Interquartile Range	Skewness	Kurtosis
	Std. Error	3,497											0,041	0,081
Dez-05	Statistic	141,77	134,12	149,42	115,56	27	46604,525	215,881	0	855	855	179	1,696	1,736
	Std. Error	3,901											0,044	0,088
Jan-06	Statistic	132,97	125,53	140,41	107,14	33	40149,183	200,373	0	825	825	161	1,794	2,264
	Std. Error	3,795											0,046	0,093
Fev-06	Statistic	125,84	119,18	132,49	100,57	33	37280,93	193,083	0	798	798	136	1,816	2,348
	Std. Error	3,395											0,043	0,086
Mar-06	Statistic	125,85	119,63	132,07	101,67	27	36822,208	191,891	0	771	771	178	1,718	1,894
	Std. Error	3,172											0,04	0,081
Abr-06	Statistic	122,99	117,34	128,64	100,19	19	35077,368	187,29	0	741	741	179	1,628	1,538
	Std. Error	2,882											0,038	0,075
Mai-06	Statistic	137,04	131,31	142,76	116,43	34	36438,164	190,888	0	710	710	186	1,489	1,046
	Std. Error	2,921											0,037	0,075
Jun-06	Statistic	131,27	125,25	137,3	111,16	56	31514,357	177,523	0	685	685	165	1,548	1,377
	Std. Error	3,071											0,042	0,085
Jul-06	Statistic	122	115,39	128,6	103,31	48	26307,143	162,195	0	656	656	183	1,558	1,466
	Std. Error	3,37											0,051	0,102
Ago-06	Statistic	117,84	111,91	123,77	100,78	33	25358,163	159,242	0	611	611	211	1,405	0,855
	Std. Error	3,024											0,046	0,093
Set-06	Statistic	115,78	109,84	121,73	99,25	32	24951,856	157,962	0	580	580	198	1,426	0,902
	Std. Error	3,032											0,047	0,094
Out-06	Statistic	117,7	112,02	123,37	103,25	32	22808,967	151,026	0	555	555	205	1,274	0,462
	Std. Error	2,894											0,047	0,094
Nov-06	Statistic	104,6	99,33	109,87	90,49	33	18735,553	136,878	0	531	531	172	1,419	0,987
	Std. Error	2,687											0,048	0,096
Dez-06	Statistic	101,35	96,03	106,68	88,2	47,5	16448,779	128,253	0	497	497	143	1,477	1,188
	Std. Error	2,715											0,052	0,104
Jan-07	Statistic	92,88	87,96	97,8	80,32	33	14640,304	120,997	0	466	466	132	1,479	1,139
	Std. Error	2,507											0,051	0,101
Fev-07	Statistic	84,91	80,72	89,1	72,83	40	12484,247	111,733	0	443	443	106	1,633	1,606
	Std. Error	2,138											0,047	0,094
Mar-07	Statistic	78,56	74,31	82,8	61,38	28	16068,933	126,763	0	1091	1091	85	2,875	11,386
	Std. Error	2,166											0,042	0,084
Abr-07	Statistic	202,2	193,01	211,39	173,16	43	79985,126	282,816	0	1089	1089	303	1,422	0,856
	Std. Error	4,688											0,041	0,081
Mai-07	Statistic	194,76	186,9	202,62	169,46	37	70169,323	264,895	0	1038	1038	352	1,296	0,432
	Std. Error	4,009											0,037	0,074
Jun-07	Statistic	210,74	202,08	219,4	186,63	76	69605,574	263,829	0	1012	1012	345	1,232	0,391
	Std. Error	4,418											0,041	0,082
Jul-07	Statistic	210,81	200,5	221,13	184,62	63	72269,248	268,829	0	978	978	330	1,264	0,464
	Std. Error	5,261											0,048	0,096
Ago-07	Statistic	189,06	179,64	198,48	165,34	44	62090,538	249,18	0	947	947	297	1,274	0,414
	Std. Error	4,805											0,047	0,094
Set-07	Statistic	188,49	178,55	198,43	165,45	56	58834,198	242,558	0	905	905	287	1,254	0,338
	Std. Error	5,07											0,051	0,102
Out-07	Statistic	181,41	172,47	190,35	158,91	80	51853,873	227,714	0	883	883	261	1,319	0,639
	Std. Error	4,56											0,049	0,098
Nov-07	Statistic	180,11	171,24	188,98	158,87	69	52222,99	228,523	0	855	855	258	1,281	0,392

Continua na página seguinte

Tabela C.1 Estatísticas da Duração da Resolução dos Defeitos. – continuação da página anterior

		Mean	Lower Bound	Upper Bound	5% Trimmed Mean	Median	Variance	Std. Deviation	Min	Max	Range	Interquartile Range	Skewness	Kurtosis
	Std. Error	4,524											0,048	0,097
Dez-07	Statistic	180,74	170,37	191,11	160,87	61	50937,27	225,693	0	827	827	276	1,189	0,088
	Std. Error	5,286											0,057	0,115
Jan-08	Statistic	153,35	145,25	161,45	131,96	45	43465,437	208,484	0	793	793	203	1,437	0,851
	Std. Error	4,13											0,048	0,097
Fev-08	Statistic	136,91	129,58	144,25	115,97	41	36831,508	191,915	0	765	765	175	1,551	1,304
	Std. Error	3,74											0,048	0,095
Mar-08	Statistic	118,21	111,86	124,56	97,56	30	31198,503	176,631	0	736	736	143	1,68	1,735
	Std. Error	3,239											0,045	0,09
Abr-08	Statistic	108,07	102,57	113,58	87,07	23	28123,051	167,699	0	706	706	134	1,796	2,269
	Std. Error	2,81											0,041	0,082
Mai-08	Statistic	113,8	108,47	119,13	95,48	17	27823,518	166,804	0	675	675	179	1,482	1,105
	Std. Error	2,718											0,04	0,08
Jun-08	Statistic	136,99	130,72	143,26	120,51	52	29499,57	171,754	0	647	647	235	1,204	0,316
	Std. Error	3,197											0,046	0,091
Jul-08	Statistic	119,3	113,23	125,38	101,49	38	24935,037	157,908	0	618	618	190	1,455	1,185
	Std. Error	3,099											0,048	0,096
Ago-08	Statistic	123,47	117,04	129,91	108,11	45	23351,872	152,813	0	582	582	219	1,232	0,515
	Std. Error	3,281											0,053	0,105
Set-08	Statistic	108,17	102,71	113,63	93,09	40	18805,708	137,134	0	558	558	191	1,363	1,059
	Std. Error	2,783											0,05	0,099
Out-08	Statistic	98,24	93,49	102,99	84,8	29	16258,529	127,509	0	522	522	177	1,337	0,873
	Std. Error	2,423											0,047	0,093
Nov-08	Statistic	95,96	91,88	100,04	82,87	51	12616,265	112,322	0	501	501	163	1,492	2,075
	Std. Error	2,083											0,045	0,091
Dez-08	Statistic	87,29	82,04	92,54	73,54	36	12864,597	113,422	0	469	469	143	1,57	1,811
	Std. Error	2,677											0,058	0,115
Jan-09	Statistic	69,8	65,75	73,84	56,41	22	9881,197	99,404	0	431	431	103	1,872	2,954
	Std. Error	2,062											0,051	0,102
Fev-09	Statistic	57,45	54,12	60,79	45,19	19	7178,125	84,724	0	400	400	83	2,115	4,358
	Std. Error	1,7											0,049	0,098
Mar-09	Statistic	58,74	55,61	61,88	46,73	21	7548,093	86,88	0	381	381	71	1,979	3,28
	Std. Error	1,599											0,045	0,09
Abr-09	Statistic	46,91	43,94	49,89	35,31	10	6111,71	78,177	0	345	345	54	2,157	3,967
	Std. Error	1,518											0,048	0,095
Mai-09	Statistic	48,15	45,16	51,15	37,8	7	6198,956	78,733	0	320	320	55	1,899	2,55
	Std. Error	1,529											0,048	0,095
Jun-09	Statistic	60,38	57,09	63,67	52,52	24	6121,753	78,242	0	290	290	84	1,398	0,789
	Std. Error	1,677											0,052	0,105
Jul-09	Statistic	47,5	44,63	50,38	39,89	16	4338,834	65,87	0	260	260	69	1,619	1,592
	Std. Error	1,466											0,054	0,109
Ago-09	Statistic	41,74	38,97	44,51	34,99	13	3487,647	59,056	0	229	229	57	1,619	1,498
	Std. Error	1,412											0,059	0,117
Set-09	Statistic	39,82	37,27	42,36	34,4	13	2833,398	53,23	0	194	194	54	1,418	0,781
	Std. Error	1,299											0,06	0,119
Out-09	Statistic	31,86	29,79	33,94	27,35	10	1881,641	43,378	0	167	167	44	1,409	0,741
	Std. Error	1,058											0,06	0,119
Nov-09	Statistic	27,6	25,77	29,43	23,96	7	1371,049	37,028	0	141	141	49	1,28	0,367

Continua na página seguinte

Tabela C.1 Estatísticas da Duração da Resolução dos Defeitos. – continuação da página anterior

		Mean	Lower Bound	Upper Bound	5% Trimmed Mean	Median	Variance	Std. Deviation	Min	Max	Range	Interquartile Range	Skewness	Kurtosis
	Std. Error	0,934											0,062	0,123
Dez-09	Statistic	22,17	20,61	23,74	19,56	8	749,375	27,375	0	110	110	39	1,198	0,323
	Std. Error	0,798											0,071	0,142
Jan-10	Statistic	13,16	12,28	14,05	11,32	5	310,417	17,619	0	76	76	18	1,419	0,854
	Std. Error	0,452											0,063	0,126
Fev-10	Statistic	8,67	8,09	9,25	7,52	3	121,781	11,035	0	52	52	14	1,37	0,976
	Std. Error	0,296											0,066	0,131
Mar-10	Statistic	2,81	2,58	3,04	2,24	1	18,133	4,258	0	24	24	4	2,007	3,929
	Std. Error	0,116											0,067	0,134

Noções Experimentais

Este capítulo do anexo pretende reforçar e ajudar a compreensão de conceitos com informação mais detalhadas sobre os mesmos e apresentar os principais tópicos presentes nesta dissertação relativos a análise estatística.

Séries Temporais Na estatística, processamento de sinal e muitos outros campos, uma série temporal é uma sequência de dados, medidos tipicamente em tempos sucessivos espaçados num intervalo de tempo que por vezes é uniforme. A análise de séries temporais engloba métodos que tentam entender a série temporal, quer para compreender o contexto subjacente dos pontos de dados (de onde vieram? porque que foram criados?) ou para fazer previsões. A previsão com séries temporais é o uso de um modelo para previsão de eventos futuros com base em eventos passados conhecidos: a previsão do futuro aponta dados antes de serem medidos. A maioria dos modelos de séries temporais podem ser descritos em termos de dois tipos básicos de componentes: tendência e sazonalidade.

Análise de Tendências Não há nenhuma técnica automática comprovada para identificar os componentes de tendência nos dados de séries temporais. No entanto, enquanto a tendência é monótona (crescente ou decrescente) a parte da análise de dados normalmente não é muito difícil. Se os dados da série temporal contém um erro considerável, então o primeiro passo no processo de identificação das tendências é chamado de nivelamento.

O nivelamento envolve sempre alguma forma de compensação local dos dados, tais que os componentes não sistemáticos das observações individuais se anulam mutuamente. A técnica mais comum é que a nivelação da média móvel, que substitui cada elemento da série quer pelos

elementos simples ou média ponderada dos n elementos circundantes, onde n é a largura da janela de nivelamento [14].

Muitos dados de séries temporais monótonas podem ser adequadamente aproximados por uma função linear, se existe um claro componente não-linear monótono, os dados precisam inicialmente de serem transformados para remover a não-linearidade. Normalmente, uma função logarítmica, exponencial, ou, menos frequentemente, polinomial pode ser usada.

Análise da Sazonalidade A sazonalidade é o outro componente geralmente presente na maioria das séries temporais. É formalmente definida como a dependência de correlação de ordem k entre cada i'ésimo elemento da série e os $(i-k)$'ésimo elemento e medido pela auto-correlação (isto é, uma correlação entre os dois termos); k é geralmente chamado de *lag* ou intervalo de tempo. Se o erro de medição não é muito grande, a sazonalidade pode ser identificada visualmente na série como um padrão que se repete a cada k elementos.

Padrões sazonais de séries temporais podem ser examinadas através de correlogramas. O correlograma (autocorrelograma) mostra graficamente e numericamente a função de auto-correlação (ACF), ou seja, os coeficientes de correlação da série (e seus erros standards) para *lags* consecutivos, num intervalo específico de *lags* (por exemplo, de 1 a 30). O gráfico de auto-correlação pode ajudar a responder a estas perguntas, entre outras:

- os dados são aleatórios?
- uma observação está relacionada com uma observação adjacente?
- a série temporal observada é ruído branco?
- a série temporal observada é sinusoidal?
- o que é um modelo apropriado para a série temporal observada?

Enquanto se analisa os correlogramas deve-se ter em mente que auto-correlações para *lags* consecutivos são formalmente dependentes. Se o primeiro elemento está muito relacionado com o segundo, e o segundo para o terceiro, então o primeiro elemento também deverá estar relacionado com o terceiro, e assim por diante. Isto implica que o padrão de dependências da série pode mudar consideravelmente após a remoção da auto-correlação (ou seja, após a diferenciação das séries).

Auto-correlações Os gráficos ou *plots* de auto-correlação são uma ferramenta usada frequentemente para verificar a aleatoriedade de um conjunto de dados. Além disso, estes gráficos são usados na fase de identificação de modelo para modelos de séries temporais auto-regressivo e de média móvel [14], nesse caso, se não se verificar a aleatoriedade, a validade de muitas das nossas conclusões estatísticas torna-se suspeita. Por isso os gráficos de auto-correlação são uma excelente forma de verificação de tal aleatoriedade.

Auto-correlações Parciais Outro método útil para analisar as dependências da série é examinar a função parcial de auto-correlação (PACF) - uma extensão de auto-correlação, onde a dependência dos elementos intermediários (aqueles dentro do *lag*) é removido. Num certo sentido, a função parcial de auto-correlação fornece uma imagem mais "limpa"das dependências da série para *lags* individuais.

Removendo as dependências da série As dependências de série para um particular *lag* k pode ser removido através da diferenciação da série, que converte cada i'ésimo elemento da série na sua diferença do $(i-k)$'ésimo elemento. Há duas razões principais para tais transformações.

Em primeiro lugar, pode-se identificar a natureza escondida das dependências sazonais na série. Auto-correlações de *lags* consecutivas são interdependentes, portanto, removendo algumas das auto-correlações mudará outras auto-correlações, ou seja, pode eliminá-las ou pode tornar uma outra sazonalidade mais aparente.

A outra razão para a remoção das dependências sazonais é tornar a série estacionária. A estacionaridade é um requisito necessário para aplicar os modelos ARIMA e outras técnicas. Numa análise de séries temporais, uma série estacionária tem uma média, variância e auto-correlação constante ao longo do tempo, o que significa que as dependências sazonais foram removidas através de diferenciação. Nesta transformação a série vai ser transformado em: $X = X - X(lag)$ e a série resultante será de comprimento $N - lag$, onde N é o comprimento da série original.

ARIMA (Auto-Regressivo Integrado Média Móvel) Os procedimentos de modelação e previsão exigem um conhecimento sobre o modelo matemático do processo. No entanto, na investigação e prática na vida real, os padrões dos dados não são claros, as observações individuais envolvem um erro considerável, e precisamos não apenas de descobrir padrões ocultos

nos dados, mas também de gerar previsões. A metodologia ARIMA explicada em [14], permite-nos fazê-lo. Esta metodologia dá-nos um grande poder e flexibilidade, mas tem o seu grau de complexidade.

O modelo geral inclui um parâmetro auto-regressivo (AR) bem como um parâmetro média móvel (MA), e explicitamente inclui na formulação do modelo a diferenciação. Especificamente, os três tipos de parâmetros no modelo são: os parâmetros auto-regressivos (p), o número de passos de diferenciação (d) e os parâmetros de média móvel (q). Nesta notação os modelos são resumidos como *ARIMA (p,d,q)*, por exemplo, um modelo descrito como (0,1,2) significa que contém 0(zero) parâmetros auto-regressivo (p) e 2 parâmetros de média móvel (q) que foram calculados após a série ter sido diferenciada uma vez (d=1).

Identificação dos Parâmetros Como mencionado anteriormente, a série a qual será aplicado o modelo ARIMA precisa de ser estacionária, ou seja, ela deve ter uma média, variância e auto-correlação constante ao longo do tempo. Para isso, geralmente a série é diferenciada até se encontrar estacionária (por vezes é necessário também aplicar uma transformação logarítmica para estabilizar a variância). O número de vezes que a série deve ser diferenciada para alcançar a estacionaridade é reflectido no parâmetro d. Para determinar o nível necessário de diferenciação, devemos examinar os gráficos dos dados e autocorrelogramas. Mudanças significativas nos gráficos dos dados e autocorrelogramas (fortes mudanças para cima ou para baixo) geralmente requerem uma diferenciação não sazonal de primeira ordem, fortes mudanças de inclinação geralmente requerem uma diferenciação não sazonal de segunda ordem. Padrões sazonais exigem uma respectiva diferenciação sazonal. Se os coeficientes de auto-correlação estimados diminuírem lentamente ao longo dos *lags*, a diferenciação de primeira ordem é normalmente necessária. No entanto, deve-se ter em mente que algumas séries podem exigir pouca ou nenhuma diferenciação, e que uma série sobre diferenciada produz estimativas dos coeficientes menos estáveis, o que significa menos rigor nas séries temporais obtidas.

Além disso, também é preciso decidir quantos parâmetros auto-regressivos (AR)(p) e de média móvel (MA)(q) são necessários para produzir um modelo eficaz, mas ainda parcimonioso, do processo (parcimonioso significa que tem o menor número de parâmetros e maior número de graus de liberdade entre todos os modelos que se ajustam aos dados). Na prática, os números dos parâmetros p ou q muito raramente precisam de ser superiores a 2.

As principais ferramentas utilizadas na fase de identificação são os *plots* ou gráficos da série, correlogramas de auto-correlação (ACF) e auto-correlação parcial (PACF). A decisão não

é simples e, em casos menos típicos exige não só experiência, mas também uma boa dose de experimentação de modelos alternativos (assim como os parâmetros técnicos do modelo ARIMA) [64]. No entanto, a maioria dos padrões empíricos de séries temporais podem ser suficientemente aproximados através de um dos cinco modelos básicos que podem ser identificados com base na forma da ACF e PACF.

O seguinte resumo é baseado em recomendações práticas de Vandaele et *al.* [7] e mais conselhos práticos a partir de McCleary et *al.* [56]. Como o número de parâmetros (a ser estimado) de cada tipo quase nunca é superior a dois, muitas vezes é prático tentar modelos alternativos sobre os mesmos dados:

- Um parâmetro auto-regressivo (p): ACF - cai exponencialmente; PACF - pico no *lag* 1, sem correlação nos outros *lags*.
- Dois parâmetros auto-regressivo (p): ACF - um padrão em forma de onda sinusoidal ou um conjunto de caídas exponenciais; PACF - picos no *lag* 1 e 2, sem correlação nos outros *lags*.
- Um parâmetro média móvel (q):ACF - pico no *lag* 1, sem correlação nos outros *lags*; PACF - cai fora de forma exponencial.
- Dois parâmetros média móvel (q): ACF - picos no *lag* 1 e 2, sem correlação nos outros *lags*.; PACF -um padrão em forma de onda sinusoidal ou um conjunto de caídas exponenciais.
- Um parâmetro auto-regressivo (p) e um parâmetro média móvel (q): ACF - caída exponencial a partir do *lag* 1; PACF - caída exponencial a partir do *lag* 1.

Estimação dos parâmetros e previsão As estimativas dos parâmetros são utilizados na fase de previsão, para calcular os novos valores da série (além daqueles que estão incluídos no conjunto de dados de entrada) e intervalos de confiança para esses valores previstos. O processo de estimação é efectuado sobre os dados transformados (diferenciado), antes das previsões serem geradas, a série tem de ser integrada (integração é o inverso da diferenciação), de modo que as previsões sejam expressas em valores compatíveis com os dados de entrada. Este recurso de integração automática é representado pela letra I, no nome da metodologia (ARIMA = Auto-Regressivo Integrado Média Móvel).

Modelos Sazonais ARIMA multiplicativo sazonal é uma generalização e extensão do método introduzido nos parágrafos anteriores para séries em que se repete um padrão de sazonalidade ao longo do tempo. Além dos parâmetros não-sazonais, é necessário estimar os parâmetros sazonais para um intervalo de tempo específico (estabelecido na fase de identificação do modelo). Análogo aos parâmetros ARIMA não sazonais, os sazonais são: auto-regressivo sazonal (ps), diferenciação sazonal (ds), e média móvel sazonal (qs). Por exemplo, o modelo (0,1,2) (0,1,1) descreve um modelo que não inclui os parâmetros auto-regressivos, dois parâmetros não- sazonais de média móvel e um parâmetro sazonal de média móvel, e esses parâmetros foram calculados para a série após ter sido diferenciada uma vez e diferenciada sazonalmente uma vez. O intervalo de tempo sazonal utilizado para os parâmetros sazonais geralmente é determinada durante a fase de identificação e deve ser explicitamente especificado.

As recomendações gerais relativas à selecção dos parâmetros a serem estimados com base na ACF e PACF também se aplicam aos modelos sazonais.

Avaliação dos Modelos Um bom modelo não deve apenas fornecer previsões suficientemente precisas, também tem de ser parcimonioso e produzir resíduos estatisticamente independentes que contêm apenas ruído e nenhum componente sistemático (por exemplo, o correlograma dos resíduos não deve revelar quaisquer dependências da série). Um bom teste ao modelo é dispor graficamente os resíduos e inspeccioná-los em busca de qualquer tendência sistemática e examinar o autocorrelograma dos resíduos (não pode existir nenhuma dependência da serie entre os resíduos). Para o modelo ARIMA ser considerado válido os resíduos devem ser distribuídos de forma sistemática em toda a série (por exemplo, poderia ser negativo na primeira parte da série e aproximar-se de zero na segunda parte). Se os resíduos contêm alguma dependência de série, provavelmente, o modelo ARIMA é inadequado. O procedimento de apreciação dos resíduos assume que qualquer resíduo resultante não são auto-correlacionados e que são normalmente distribuídos.

A figura D.1 mostra os gráficos necessários para a validação do modelo ARIMA. No nosso exemplo, apenas o gráfico do canto superior esquerdo válida o nosso modelo. No gráfico do canto superior direito, o teste à aleatoriedade é violada, nos gráficos inferiores o teste à distribuição normal dos resíduos é violado. No histograma é apresentada uma distribuição em forma de U e o gráfico ao lado mostra que a distribuição não é normal.

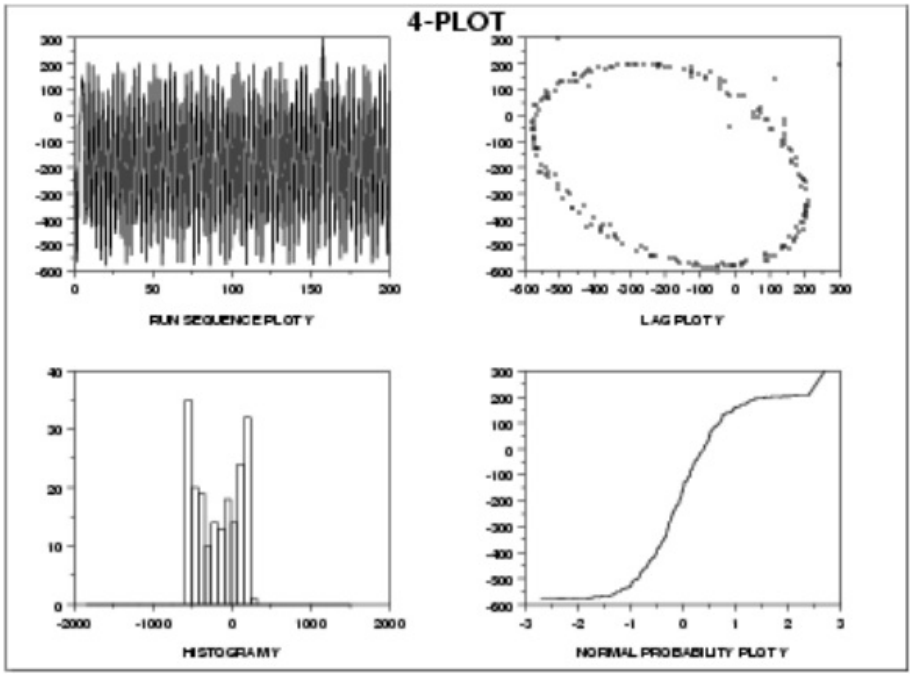


Figura D.1 Gráficos para a validação do modelo - Modelo ARIMA inválido. Adaptado de [16].

Bibliografia

[1] Ieee standard for software maintenance. *IEEE Std 1219-1998*, 1998.

[2] Industry implementation of international standard iso/iec 12207: 1995. (iso/iec 12207 standard for information technology - software life cycle processes - implementation considerations. *IEEE/EIA 12207.2-1997*, 1998.

[3] International standard - iso/iec 14764 ieee std 14764-1999 software engineering - software maintenance. *ISO/IEC 14764:1999 (E) IEEE Std 14764-1999*, 1999.

[4] Proceedings international symposium on principles of software evolution. In *Principles of Software Evolution, 2000. Proceedings. International Symposium on*, 2000.

[5] International standard - iso/iec 14764 ieee std 14764-2006 software engineering - software life cycle processes - maintenance. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, 2006.

[6] Iwpse-evol '09: Proceedings of the joint international and annual ercim workshops on principles of software evolution (iwpse) and software evolution (evol) workshops, 2009. General Chair-Mens, Tom and Program Chair-Mens, Kim and Program Chair-Wermelinger, Michel.

[7] S. Anderson, A. Auquier, WW Hauck, D. Oakes, W. Vandaele, and H.I. Weisberg. Statistical methods for comparative studies. 1980.

[8] E. J. Barry, C. F. Kemerer, and S. A. Slaughter. How software process automation affects software evolution: a longitudinal empirical analysis: Research articles. *J. Softw. Maint. Evol.*, 19(1):1–31, 2007.

[9] A. Bauer and M. Pizka. The contribution of free software to software evolution. In *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of*, pages 170–179, Sept. 2003.

[10] K. H. Bennett and V. T. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 73–87, New York, NY, USA, 2000. ACM.

[11] A. Bernstein, J. Ekanayake, and M. Pinzger. Improving defect prediction using temporal features and non linear models. In *IWPSE '07: Ninth international workshop on Principles of software evolution*, pages 11–18, New York, NY, USA, 2007. ACM.

[12] J. Bevan, E. J. Whitehead Jr., S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 177–186, New York, NY, USA, 2005. ACM.

[13] S. Biyani and P. Santhanam. Exploring defect data from development and customer usage on software modules over multiple releases. In *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, pages 316–320, Nov 1998.

[14] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1994.

[15] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel. Towards a taxonomy of software change: Research articles. *J. Softw. Maint. Evol.*, 17(5):309–332, 2005.

[16] J. Caldeira. Information technology service management: An experimental approach towards it service prediction. Master's thesis, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, 2009.

[17] R.G. Canning. That maintenance ?iceberg? *EDP Analyzer*, 10(10):1–14, 1972.

[18] A. Capiluppi, M. Morisio, and J. F. Ramil. The evolution of source folder structure in actively evolved open source systems. In *Software Metrics, 2004. Proceedings. 10th International Symposium on*, pages 2–13, Sept. 2004.

[19] A. F. Chalmers. *What is this thing called science?* Hackett Publishing Company, 1999.

[20] N. Chapin, J. E. Hale, K. M. Kham, J. F. Ramil, and W. Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance*, 13(1):3–30, 2001.

[21] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, New York, NY, USA, 2003. ACM.

[22] S. Cook, R. Harrison, M. M. Lehman, and P. Wernick. Evolution in software systems: foundations of the spe classification scheme: Research articles. *J. Softw. Maint. Evol.*, 18(1):1–35, 2006.

[23] M. D'Ambros. Supporting software evolution analysis with historical dependencies and defect information. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 412–415, 28 2008-Oct. 4 2008.

[24] M. D'Ambros and M. Lanza. Bugcrawler: Visualizing evolving software systems. In *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, pages 333–334, 2007.

[25] M. D'Ambros and M. Lanza. A flexible framework to support collaborative software evolution analysis. In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, pages 3–12, April 2008.

[26] M. D'Ambros and M. Lanza. Visual software evolution reconstruction. *J. Softw. Maint. Evol.*, 21(3):217–232, 2009.

[27] M. D'Ambros, M. Lanza, and M. Lungu. The evolution radar: visualizing integrated logical coupling information. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 26–32, New York, NY, USA, 2006. ACM.

[28] M. D'Ambros, M. Lanza, and M. Lungu. Visualizing co-change information with the evolution radar. *Software Engineering, IEEE Transactions on*, 35(5):720–735, 2009.

[29] M. D'Ambros, M. Lanza, and M. Pinzger. The metabase: Generating object persistency using meta descriptions. *Proceedings of FAMOOSR*, 2007, 2007.

[30] S. C. Eick, J. L. Steffen, and E. E. Sumner Jr. Seesoft-a tool for visualizing line oriented software statistics. *Software Engineering, IEEE Transactions on*, 18(11):957–968, Nov 1992.

[31] M. Fischer and H. Gall. Evograph: A lightweight approach to evolutionary and structural analysis of large software systems. In *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*, pages 179–188, Oct. 2006.

[32] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 23–32, Sept. 2003.

[33] E. Fuentetaja and D. J. Bagert. Software evolution from a time-series perspective. In *Software Maintenance, 2002. Proceedings. International Conference on*, pages 226–229, 2002.

[34] D. M. German, A. Hindle, and N. Jordan. Visualizing the evolution of software using softChange. *International Journal of Software Engineering and Knowledge Engineering*, 16(1):5–21, 2006.

[35] M. W. Godfrey and D. M. German. The past, present, and future of software evolution. In *Frontiers of Software Maintenance, FoSM, 2008.*, pages 129–138, 2008.

[36] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, Washington, DC, USA, 2000. IEEE Computer Society.

[37] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. An empirical study of global software development: distance and speed. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 81–90, Washington, DC, USA, 2001. IEEE Computer Society.

[38] C. F. Kemerer and S. Slaughter. An empirical approach to studying software evolution. *Software Engineering, IEEE Transactions on*, 25(4):493–509, Jul/Aug 1999.

[39] M. Kläs, H. Nakao, F. Elberzhager, and J. Münch. Support planning and controlling of early quality assurance by combining expert judgment and defect data - a case study. In *ESE '10: Empirical Software Engineering, Vol.15*, pages 423–454. Springer, 2010.

[40] T. S. Kuhn. *The structure of scientific revolutions*. University of Chicago Press Chicago, 1970.

[41] M. Lanza. The evolution matrix: recovering software evolution using software visualization techniques. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 37–42, New York, NY, USA, 2001. ACM.

[42] M. Lanza and S. Ducasse. Polymetric views - a lightweight visual approach to reverse engineering. *Software Engineering, IEEE Transactions on*, 29(9):782–795, Sept. 2003.

[43] M. M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, Sept. 1980.

[44] M. M. Lehman. Software uncertainty and the role of case in its minimisation and control. In *Information Technology, 1990. 'Next Decade in Information Technology', Proceedings of the 5th Jerusalem Conference on (Cat. No.90TH0326-9)*, pages 236–246, Oct 1990.

[45] M. M. Lehman. Uncertainty in computer application is certain-software engineering as a control. In *CompEuro '90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pages 468–474, May 1990.

[46] M. M. Lehman. Laws of software evolution revisited. In *Software Process Technology*, pages 108–124, 1996. 10.1007/BFb0017737.

[47] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[48] M. M. Lehman and J. F. Ramil. Rules and tools for software evolution planning and management. *Ann. Softw. Eng.*, 11(1):15–44, 2001.

[49] M. M. Lehman and J. F. Ramil. Software evolution and software evolution processes. *Ann. Softw. Eng.*, 14(1-4):275–309, 2002.

[50] M. M. Lehman and J. F. Ramil. Software evolution: background, theory, practice. *Inf. Process. Lett.*, 88(1-2):33–44, 2003.

[51] M. M. Lehman, J. F. Ramil, and G. Kahen. Evolution as a noun and evolution as a verb. In *Proceedings of the two day workshop on software and business co-evolution (SOCE'2000)*, 2000.

[52] B. P. Lientz and E. B. Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.

[53] M. Lungu, M. Lanza, and T. Gîrba. The small project observatory. In *1st International Workshop on Academic Software Development Tools and Techniques (WASDeTT-1)*, 2008.

[54] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, In Press, Corrected Proof, 2009.

[55] N. H. Madhavji, J. F. Ramil, and D. Perry. *Software Evolution and Feedback: Theory and Practice*. John Wiley & Sons, 2006.

[56] R. McCleary, R. Hay, E.E. Meidinger, D. McDowall, and K.C. Land. *Applied time series analysis for the social sciences*. Sage Publications Beverly Hills, CA, 1980.

[57] T. Mens, J. Buckley, M. Zenger, and A. Rashid. Towards a taxonomy of software evolution. In *Proc. Workshop on Unanticipated Software Evolution*. Citeseer, 2003.

[58] T. Mens and S. Demeyer. *Software Evolution*. Springer Publishing Company, Incorporated, 2008.

[59] T. Mens, J. F. Ramil, and S. Degrandsart. The evolution of eclipse. In *Software Maintenance, ICSM 2008. IEEE International Conference on*, pages 386–395, 2008.

[60] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in software evolution. In *Principles of Software Evolution, Eighth International Workshop on*, pages 13–22, Sept. 2005.

[61] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 452–461, New York, NY, USA, 2006. ACM.

[62] M. Ogawa and K. Ma. code swarm: A design study in organic software visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1097–1104, 2009.

[63] L. D. Panjer. Predicting eclipse bug lifetimes. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, Washington, DC, USA, 2007. IEEE Computer Society.

[64] A. Pankratz. Forecasting with univariate Box-Jenkins models: Concepts and cases. *JOHN WILEY & SONS, INC., 605 THIRD AVE., NEW YORK, NY 10158, USA, 1983, 560*, 1983.

[65] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 67–75, New York, NY, USA, 2005. ACM.

[66] M. Pizka and A. Bauer. A brief top-down and bottom-up philosophy on software evolution. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*, pages 131–136, Sept. 2004.

[67] M. Polo, M. Piattini, F. Ruiz, and C. Calero. MANTEMA: A complete rigorous methodology for supporting maintenance based on the ISO/IEC 12207 Standard. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 1999.

[68] U. Raja, D. P. Hale, and J. E. Hale. Modeling software evolution defects: a time series approach. *J. Softw. Maint. Evol.*, 21(1):49–71, 2009.

[69] V. T. Rajlich and K. H. Bennett. A staged model for the software life cycle. *Computer*, 33(7):66–71, 2000.

[70] J. Ratzinger, M. Fischer, and H. Gall. Evolens: lens-view visualizations of evolution data. In *Principles of Software Evolution, Eighth International Workshop on*, pages 103–112, Sept. 2005.

[71] W. Scacchi. Understanding open source software evolution. In N. H. Madhavji, J. Fernandez-Ramil, and D. Perry, editors, *Software Evolution and Feedback: Theory and Practice*, pages 181–206, 2006.

[72] A. Schröter, T. Zimmermann, and A. Zeller. Predicting component failures at design time. In *ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, pages 18–27, New York, NY, USA, 2006. ACM.

[73] H. Siy, P. Chundi, D. J. Rosenkrantz, and M. Subramaniam. Discovering dynamic developer relationships from software version histories by time series segmentation. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 415–424, Oct. 2007.

[74] E. B. Swanson. The dimensions of maintenance. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 492–497, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[75] N. Tsantalis. Predicting the probability of change in object-oriented systems. *IEEE Trans. Softw. Eng.*, 31(7):601–614, 2005. Member-Chatzigeorgiou, Alexander and Member-Stephanides, George.

[76] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: a project memory for software development. *Software Engineering, IEEE Transactions on*, 31(6):446–465, June 2005.

[77] L. Voinea and A. Telea. Cvsgrab: Mining the history of large software projects. In *Proc. EuroVis'06*. IEEE Computer Society Press, 2006.

[78] L. Voinea and A. Telea. Visual analytics: Visual data mining and analysis of software repositories. *Comput. Graph.*, 31(3):410–428, 2007.

[79] L. Voinea and A. Telea. Visual querying and analysis of large software repositories. *Empirical Softw. Engg.*, 14(3):316–340, 2009.

[80] L. Voinea, A. Telea, and J. J. van Wijk. Cvsscan: visualization of code evolution. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 47–56, New York, NY, USA, 2005. ACM.

[81] M. Wermelinger, Y. Yu, and A. Lozano. Design principles in architectural evolution: A case study. In *Software Maintenance, ICSM 2008. IEEE International Conference on*, pages 396–405, 2008.

[82] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 531–540, New York, NY, USA, 2008. ACM.

[83] T. Zimmermann and N. Nagappan. Predicting Defects with Program Dependencies. In *ESEM '09: 3th International Symposium on Empirical Software Engineering and Measurement*, 2009.

[84] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, Washington, DC, USA, 2007. IEEE Computer Society.