

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

CrystalBox - uma ferramenta para documentação ágil de software

Nuno Miguel Moreira Baldaia de Queirós

Mestrado em Engenharia Informática

Orientador: Ademar Aguiar (Prof. Doutor)

Novembro de 2009

Resumo

A documentação tem um papel fundamental no desenvolvimento de software. Grande parte do esforço de desenvolvimento é gasto na formalização da informação, i.e., na leitura e compreensão de requisitos, especificações informais, desenhos, memorandos, assim como outros documentos informais, com o objectivo de produzir código fonte concreto e modelos.

Embora os métodos tradicionais utilizem frequentemente uma grande quantidade de documentação para garantir o nível exigido de profissionalismo, disciplina e compreensão num projecto, os métodos ágeis são geralmente muito restritivos no que diz respeito à documentação, considerando que esta pode ser difícil, cara e cansativa, especialmente quando não suportada por ferramentas e métodos adequados.

Apesar da prática ágil da “não documentação” ser bem sucedida em alguns projectos ágeis, tal não é necessariamente verdade para todos eles, nomeadamente para projectos grandes e complexos. Trabalhos recentes que abordam os problemas de documentação em processos ágeis revelam uma atitude não tão extrema para o assunto e sugerem que a documentação “deve ser concisa e objectiva”, i.e., “ágil”.

No entanto, a documentação ágil é ainda frequentemente ignorada, principalmente pelos seguintes motivos: o seu valor não tem efeito imediato; não é uma actividade apreciada; e há ainda falta de ferramentas adequadas para apoiar de uma forma fácil, rápida e sustentável.

Esta dissertação foca-se essencialmente no desenvolvimento de uma ferramenta simples para suporte à documentação ágil de projectos de software, com uma especial atenção ao desenvolvimento de software *open-source*.

Este trabalho propõe uma ferramenta baseada na web, chamada CrystalBox, cujos principais objectivos são os de ser uma ferramenta flexível, de fácil adopção, simples de usar e que permita a criação colaborativa de documentação de software em sintonia com todas as outras actividades importantes do desenvolvimento de software.

A ferramenta CrystalBox oferece um importante incentivo à documentação, ao disponibilizar um mecanismo de transclusão para incluir artefactos de software em documentos, a fim de produzir documentação semanticamente consistente.

A ferramenta foi avaliada e testada com base em três casos de estudo que deram um importante feedback sobre as verdadeiras capacidades da ferramenta e suas vantagens no contexto para o qual foi desenvolvida.

Abstract

Documentation plays a central role in software development. Most of the development effort is spent on formalizing information, i.e., on reading and understanding requirements, informal specifications, drawings, memos, and other informal documents, in order to produce concrete source code and models.

While traditional methods often use a lot of documentation to ensure the required level of skill, discipline, and understanding in a project, agile methods are generally very restrictive with regarding to documentation, considering that it can be hard, costly, and tiresome to do, especially when not supported by appropriate tools and methods.

Despite the agile practice of “no documentation” leading to success in some agile projects, this is not necessarily true to all of them, namely large and complex ones. Recent work addressing the problems of documentation on agile processes reveals a not so extreme attitude towards the subject and suggests that documentation “should be lean and mean”, i.e., “agile”.

However, agile documentation is still commonly discarded due mainly for the following reasons: its value has no immediate effect; it is not an appreciated activity; and there is still lacking convenient tools to support it in an easy, quick and sustainable way.

This dissertation focuses mainly on the development of a simple and effective tool to support agile documentation of software projects, with a special attention to open-source software development.

This work proposes a web based tool, named CrystalBox, which main goals are to be flexible, easily adoptable, simple to use, and to allow the collaborative authoring of software documentation in sync with all the other important software development activities.

CrystalBox offers an important incentive to document, using a lightweight transclusion mechanism to include software artifacts into documents in order to produce semantically consistent documentation.

The tool was evaluated and tested with the help of three case studies that gave an important feedback of the real capabilities of the tool, and its advantages in the context for which it was developed.

Agradecimentos

Agradeço ao Prof. Ademar Aguiar por todo o conhecimento que me transmitiu, por todo o apoio que me prestou e pela confiança que depositou em mim para a elaboração desta dissertação.

Agradeço ao Prof. Eugénio Oliveira, por toda a atenção dedicada.

Agradeço aos meus pais tudo o que me deram, que considero ser muito mais do que o necessário para concluir esta dissertação.

Agradeço à minha irmã por ter ajudado tanto os meus pais a dar-me tudo o que tenho.

Agradeço aos pais Leitão, simplesmente pelo facto de o serem, e por me terem apoiado e incentivado a terminar esta dissertação.

Agradeço à Susana parte do que sou. Agradeço-lhe também por me ter acompanhado e apoiado durante vários anos no meu percurso académico e profissional e por me ter dado todo o apoio para que tenha sido possível a conclusão desta dissertação.

Agradeço à Raquel por se rir tanto.

Agradeço ao Tó por ser astronauta.

Agradeço ao manjerico da Susana por se ter aguentado vivo até à entrega desta dissertação.

Agradeço a todos os outros por fazerem parte da minha vida.

Nuno Baldaia

“Do the simplest thing that could possibly work”

Kent Beck

Conteúdo

Resumo	i
Abstract	iii
1 Introdução	1
1.1 Documentação ágil de software	1
1.2 Objectivos da investigação	2
1.3 Principais resultados	3
1.4 Organização do documento	3
2 Revisão bibliográfica	5
2.1 Técnicas de documentação de software	5
2.1.1 Literate Programming	6
2.1.2 Métodos <i>single-source</i>	8
2.1.3 Métodos <i>multiple-source</i>	9
2.2 Metodologias ágeis de desenvolvimento de software	15
2.2.1 Valores	16
2.2.2 Princípios	17
2.3 Documentação de software e agilidade	17
2.3.1 Modelos, documentos e código-fonte	19
2.3.2 Circunstâncias que justificam a documentação	19
2.3.3 Características de um documento ágil	20
2.3.4 Questões relacionadas com a documentação	21
2.3.5 Boas práticas na documentação de software	24
2.3.6 Algumas práticas de Extreme Programming	25
3 Problema	27
3.1 Definição do problema	28
3.2 Justificação do problema	28
3.2.1 Literate Programming	29
3.2.2 Métodos <i>single-source</i>	29
3.2.3 Métodos <i>multiple-source</i>	30
3.3 Resultados esperados	32
3.4 Método de investigação	33
4 Projectos exploratórios	35
4.1 XSDoc for SnipSnap	36
4.2 XSDoc for MoinMoin	39
4.3 Adaptação do XSDoc a outros wikis	42

5	CrystalBox	45
5.1	Objectivos	45
5.2	Requisitos	46
5.3	Arquitectura lógica e física	46
5.3.1	Camada de dados	47
5.3.2	Camada aplicacional	47
5.3.3	Cliente	48
5.4	Produto	48
5.4.1	CrystalBox BaseStation	48
5.4.2	CrystalBox Client	51
5.5	Arquitectura tecnológica	54
5.5.1	Ruby on Rails	54
5.5.2	CrystalBox BaseStation	56
5.5.3	CrystalBox Client	62
5.6	Mecanismos de evolução	67
5.6.1	Suporte para novos recursos	67
5.6.2	Suporte para acesso a novos sistemas de controlo de versões	68
5.7	Considerações finais	69
6	Validação	71
6.1	Caso de estudo 1 – CrystalBox BaseStation	71
6.2	Caso de estudo 2 – Disciplina MIEIC/LDSO-2009/10	74
6.3	Caso de estudo 3 – Plataforma web escolinhas.pt	74
6.4	Considerações finais	76
7	Conclusões e trabalho futuro	77
7.1	Problemas identificados	78
7.2	Trabalho futuro	78
	Referências	81

Lista de Figuras

2.1	Literate Programming: <i>tangling</i> e <i>weaving</i> de documentos (Aguilar, 2003)	7
2.2	Valores do manifesto da Aliança Ágil (Agile Alliance, 2001)	16
2.3	Relação entre modelos, documentos, código-fonte e documentação (Ambler, 2003a)	19
4.1	Sistema de gestão de utilizadores para o SnipSnap (FEUP/DEEC, 2006c)	36
4.2	Configuração de um projecto de documentação no SnipSnap (FEUP/DEEC, 2006c)	37
4.3	Macro <i>source</i> no wiki SnipSnap (FEUP/DEEC, 2006c)	38
4.4	Macro <i>javaml-uml</i> no wiki SnipSnap (FEUP/DEEC, 2006c)	39
4.5	Configuração de um projecto de documentação no MoinMoin (FEUP/DEEC, 2006b)	40
4.6	Vista <i>sourcecode</i> da <i>Resource Macro</i> do MoinMoin (FEUP/DEEC, 2006b)	41
4.7	Exemplo da utilização da vista <i>uml</i> da macro <i>Resource Macro</i> do MoinMoin (FEUP/DEEC, 2006b)	42
4.8	Arquitectura utilizada nos projectos exploratórios <i>XSDoc for SnipSnap</i> e <i>XSDoc for MoinMoin</i>	42
4.9	Arquitectura para adaptação do XSDoc aos vários wikis	43
5.1	Principais casos de utilização da ferramenta CrystalBox	47
5.2	Arquitectura física da ferramenta CrystalBox	48
5.3	CrystalBox BaseStation – Registo de utilizadores	49
5.4	CrystalBox BaseStation – Lista de projectos	50
5.5	CrystalBox BaseStation – Configuração do sistema de controlo de versões	51
5.6	CrystalBox Client – visualização de um recurso na caixa CrystalBox	53
5.7	CrystalBox Client – visualização de um recurso incluído no corpo documento	54
5.8	Exemplo de utilização da macro CrystalBox no wiki do Trac	55
5.9	CrystalBox BaseStation – Diagrama de classes do sistema de configuração	56
5.10	CrystalBox BaseStation – Diagrama de classes dos mecanismos de acesso a sistemas de controlo de versões da aplicação	57
5.11	CrystalBox BaseStation – Diagrama de classes dos mecanismos de processamento de recursos da aplicação	58
5.12	Tentativa de acesso a recursos disponíveis na aplicação CrystalBox BaseStation utilizando o objecto XMLHttpRequest	63
5.13	Hipotética utilização do objecto JSONRequest para acesso a recursos disponíveis na aplicação CrystalBox BaseStation	64
5.14	Acesso a recursos da CrystalBox BaseStation por JSONP	66
6.1	Configuração do projecto para documentação da CrystalBox	72
6.2	Evolução de um recurso CrystalBox referenciado ao longo do tempo	73

6.3	Documentação produzida com a CrystalBox	74
6.4	Documentação com recurso à CrystalBox na disciplina MIEIC/LDSO-2009/10 . .	75
6.5	Documentação com recurso à CrystalBox da plataforma web escolinhas.pt	76

Abreviaturas e Símbolos

API	Application Programming Interface
AST	Abstract Syntax Tree
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
REST	Representational State Transfer
ROI	Return on Investment
TCO	Total Cost of Ownership
URL	Uniform Resource Locator
VCS	Version Control System
XSS	Cross-site Scripting

Capítulo 1

Introdução

A documentação constitui uma actividade importante no desenvolvimento de software. Grande parte do esforço de desenvolvimento é gasto na formalização da informação, ou seja, na leitura e compreensão de requisitos, especificações informais, desenhos, memorandos e outros documentos informais, a fim de produzir modelos e código-fonte concreto.

Embora os métodos tradicionais de desenvolvimento de software usem frequentemente uma grande quantidade de documentação para garantir o nível exigido de profissionalismo, disciplina e compreensão de um projecto, os métodos ágeis são geralmente muito restritivos no que diz respeito à produção de documentação, considerando-a difícil, cara e cansativa, especialmente quando não suportada por ferramentas e métodos adequados.

Apesar da prática ágil da “não documentação” ser bem sucedida em alguns projectos ágeis, tal não é necessariamente verdade para projectos de grande dimensão e complexidade. Trabalhos recentes relativos à abordagem dos problemas de documentação sobre processos ágeis demonstram uma atitude não tão extrema para o assunto, e sugerem práticas e recomendações para a produção de documentação ágil de software.

Nesta dissertação são analisadas técnicas e ferramentas de documentação de software, e é proposta uma nova ferramenta com o objectivo de facilitar e incentivar a documentação ágil de software, assim como a documentação de software em geral.

1.1 Documentação ágil de software

O objectivo fundamental de um processo é o de melhorar a qualidade e reduzir os custos, o que muitas vezes é alcançável pela formalização do processo, a mecanização adequada das actividades humanas e automatização de tarefas repetitivas. Embora essas ideias se adequem bem a muitas actividades de engenharia, não são muito eficazes para actividades intensivamente criativas, como o design de software, programação e funções de engenharia de software (Aguilar, 2003).

Os processos tradicionais utilizam demasiada formalidade e uma grande quantidade de documentação, como meio para resolver a falta de habilidade, disciplina e compreensão de equipas de software. Apesar dos seus problemas (Highsmith, 2000), este tipo de processos são úteis em

ambientes de desenvolvimento, onde os processos mais flexíveis não se encaixam bem, como as grandes organizações, grandes equipas com baixo nível de competências técnicas, ou equipas com problemas de comunicação.

Os processos ágeis (Agile Alliance, 2001), por outro lado, assentam na adopção de práticas simples, mas eficazes, que ajudam a melhorar a qualificação e produtividade da equipa em vez de promoverem a disciplina e formalidade excessivas e a melhorar a comunicação da equipa como alternativa à produção de documentação em excesso.

Embora o objectivo extremo da “não documentação” inicialmente proclamado pelas metodologias ágeis possa ser bem sucedido em alguns projectos, tal não é necessariamente verdade para todos eles. Para projectos distribuídos, de grande dimensão, complexidade, ou para o desenvolvimento de sistemas reutilizáveis (por exemplo, componentes, frameworks aplicativos, ou linhas de produto), o sucesso depende geralmente de uma boa documentação. Por exemplo, o JUnit (Beck and Gamma, 2009), uma framework simples e muito bem sucedida para testes unitários, desenvolvida por praticantes de Extreme Programming (XP) (Beck, 2000), é acompanhada de documentação simples e eficaz. Os agilistas reconhecem que, para certos casos, uma boa documentação pode ser valiosa, mas esse valor deve ser constantemente contrabalançado com o esforço necessário para a produzir.

Trabalhos recentes sobre documentação de projectos ágeis sugerem que em vez da prática extrema da “não documentação” a documentação deve ser “concisa e objectiva”, ou seja, “ágil”, com a finalidade de “ter documentação suficiente apenas no momento certo, e apenas para o público certo” (Ambler, 2003a; Rueping, 2003).

Ademar Aguiar (Aguiar, 2003) defende que a resistência ainda existente à prática da documentação nos processos ágeis se deve, em parte, à falta de ferramentas adequadas que suportem a sua autoria de forma fácil, expedita e eficaz, ou seja, “ágil”.

1.2 Objectivos da investigação

A melhor altura para documentar a compreensão de um sistema de software é no momento em que este está a ser desenhado e/ou programado, altura em que todo o processo mental e criativo está presente nos autores do sistema. Essa compreensão pode ser documentada internamente no código-fonte, ou externamente em documentos.

A *documentação interna* é feita geralmente sob a forma de comentários no próprio código-fonte, tendo como propósito comunicar as intenções de fragmentos de código tais como classes, métodos ou atributos. Este tipo de documentação fica, no entanto, limitada à documentação de baixo nível dos sistemas.

A *documentação externa*, pelo contrário, tem como propósito a documentação de alto nível dos sistemas, tal como a descrição dos seus componentes, arquitecturas e suas interligações. Para tal, recorre geralmente aos diversos artefactos produzidos em projectos de software, tais como documentos, código-fonte e modelos. Como desvantagem, este tipo de documentação requer um

esforço significativo para a preservação da consistência semântica entre artefactos, uma vez estes estarem sujeitos a alterações durante todo o ciclo de vida de um projecto.

Esta dissertação tem como objectivo principal a concepção e execução de uma ferramenta que facilite e promova a escrita de documentação externa de software em suporte web de forma expedita e sustentável. Para tal, deverá garantir a consistência semântica entre artefactos utilizados, ser suficientemente abrangente relativamente ao tipo de projectos para os quais possa ser adoptada (devendo, para tal, ser facilmente extensível), e ser de fácil adopção e utilização.

Desta forma, o trabalho desenvolvido no âmbito desta dissertação procura promover a prática da documentação em projectos ágeis de software durante todo o ciclo de desenvolvimento, apostando, para tal, na facilidade de adopção e utilização da ferramenta proposta e na potencialidade da garantia da consistência semântica entre artefactos utilizados, o que poderá ser um forte incentivo à documentação por reduzir significativamente a relação custo/benefício na produção de documentação.

Muito embora o objectivo de investigação desta dissertação tenha como foco principal a documentação de projectos ágeis, e em especial o desenvolvimento *open-source*, a solução proposta poderá igualmente ser útil para a documentação de software em geral.

1.3 Principais resultados

Uma análise detalhada das principais técnicas de documentação de software existentes, exploradas em detalhe na Secção 2.1, permitiu reunir uma série de requisitos e objectivos que levaram ao conceito de uma nova ferramenta independente, com base na web, denominada de CrystalBox, para apoio à documentação de software.

A solução proposta foi concretizada através do desenvolvimento da ferramenta, o que permitiu já comprovar o conceito proposto com base em três casos de estudo reais em contextos académicos e empresariais.

Os principais objectivos delineados para esta dissertação foram cumpridos com êxito. A facilidade de integração da ferramenta desenvolvida com ferramentas de autoria de base web, e a possibilidade de utilização de artefactos nos documentos produzidos de forma extremamente fácil e consistente com o projecto de software (facilitando a manutenção da documentação durante todo o ciclo de desenvolvimento), constituíram um forte incentivo à documentação de projectos ágeis uma vez permitirem uma redução significativa no custo da produção e manutenção da documentação.

O conceito e produto desenvolvidos formam ainda uma base interessante que sugere diversas evoluções futuras.

1.4 Organização do documento

Esta dissertação está dividida em sete capítulos, constituindo, o presente capítulo, uma introdução geral ao documento.

No Capítulo 2 é feita uma revisão do estado da arte das principais técnicas de documentação de software. É feita uma breve abordagem às metodologias ágeis de desenvolvimento de software e são abordados os problemas da documentação no contexto destas metodologias.

No Capítulo 3 é feita a descrição do problema identificado no âmbito deste trabalho através de uma análise detalhada das técnicas de documentação de software existentes descritas na Seção 2.1. Neste capítulo são também descritos os objectivos do trabalho e o método de investigação utilizado.

No Capítulo 4 são descritos dois projectos exploratórios que estiveram na origem deste trabalho, tendo fornecido experiências valiosas para a definição do problema aqui abordado.

No Capítulo 5 é feita uma descrição detalhada da solução proposta para o problema definido: a ferramenta de documentação de software CrystalBox.

No Capítulo 6 é apresentada a validação da solução proposta através de três casos de estudo utilizando a ferramenta desenvolvida.

No Capítulo 7 são apresentadas as conclusões, os problemas identificados e o possível trabalho futuro de evolução da solução apresentada.

Capítulo 2

Revisão bibliográfica

Neste capítulo é feita uma revisão bibliográfica sobre as principais técnicas de documentação de software, sobre as metodologias ágeis de desenvolvimento de software e ainda sobre a documentação de software no contexto destas metodologias.

2.1 Técnicas de documentação de software

O desenvolvimento de software é uma actividade exigente. Os programadores mapeiam constantemente a sua representação mental de uma solução para outras representações usáveis pelos computadores. Adicionalmente, a compreensão relativa às soluções desenvolvidas deve ser preservada na forma de documentos que podem ser destinados a diferentes utilizadores e, como tal, devem ser de diferentes formas e requisitos.

De forma geral, os programadores produzem documentos recorrendo a técnicas e ferramentas que não satisfazem por completo as suas necessidades, o que acaba por introduzir novas dificuldades a todo o processo de desenvolvimento. Pelo facto de a documentação ser uma actividade custosa e os programadores não verem nela benefícios imediatos, estes tendem a focar-se apenas na programação, desprezando quase por completo a documentação.

De entre os vários artefactos produzidos durante o desenvolvimento de software, destacam-se o *código-fonte*, *modelos* e *documentos*. Tais artefactos devem ser constantemente revistos e modificados para que seja preservada a sua consistência durante todo o ciclo de desenvolvimento.

A compreensão de sistemas de software pode ser documentada internamente no código-fonte, ou externamente em documentos.

Documentação interna A documentação interna é feita geralmente em comentários no código-fonte, sendo a sua consistência preservada de forma natural ao longo do tempo. A documentação interna fica, no entanto, limitada à documentação de baixo nível relativamente às intenções de fragmentos de código tais como classes, métodos ou atributos, sendo desadequada para a documentação transversal de alto nível.

Documentação externa O recurso a documentos externos permite a escrita de documentação de alto nível, descrevendo os componentes e interligações de uma arquitectura, ou interacções

entre classes, utilizando, para tal, artefactos como documentos, código-fonte e modelos. No entanto, a preservação da consistência semântica entre artefactos e documentação produzida é, geralmente, um problema a considerar, uma vez que tais artefactos estão sujeitos a alterações durante todo o tempo de vida de evolução de um projecto.

No presente capítulo são abordadas algumas técnicas de documentação de software.

2.1.1 Literate Programming

Literate Programming (Knuth, 1984) é uma técnica de documentação de software criada por Donald Knuth em 1984 que sugere uma abordagem e ferramentas para resolver o problema de manter código-fonte e documentação de forma consistente e consolidada ao longo do tempo.

“Vamos mudar a nossa atitude tradicional na construção de programas: em vez de imaginar que a nossa tarefa principal é instruir o computador para o que fazer, vamos concentrar-nos em explicar a seres humanos o que queremos que o computador faça.”

(Knuth, 1984)

A técnica envolve a escrita de documentação e código num só documento fonte, organizado psicologicamente de forma a ser compreendido por pessoas e não por computadores. A técnica tem as seguintes características distintivas:

Verosimilhança O código e documentação são escritos juntos num mesmo documento, o que assegura que a documentação possa evoluir independentemente do código, mas sempre em consistência com ele.

Disposição psicológica Um programa pode ser organizado de uma forma considerada mais apropriada para a compreensão humana, sem ter obrigatoriamente que seguir a estrutura esperada pelo computador.

Fácil leitura O autor sugere considerar programas como obras de literatura, fáceis de ler e compreender, que incluam referências cruzadas e índices, e que se pareçam tal como livros devidamente formatados.

Um programa literate é um único documento fonte que contém fragmentos de programas e texto explicativo, organizado em pequenas secções, dispostas pela ordem pela qual o autor entender ser a melhor para descrever o programa. Esta combinação não se trata de código-fonte, nem de documentação, e é denominada tradicionalmente por *documento web*, proveniente da noção de teia de informação, e nada tem a ver com a World Wide Web que surgiu alguns anos mais tarde.

Os programas literate podem incluir tanto documentação interna, como documentação externa, podendo incluir não apenas a descrição do propósito funcional do código, mas também informação de alto nível, tal como definição de problemas, decisões de design, manuais de utilização, ou qualquer outra informação importante para melhorar a compreensão de um programa.

Os sistemas de Literate Programming, tais como o WEB (Knuth, 1983), o CWEB (Knuth and Levy, 1994), ou o noweb (Ramsey, 1994), disponibilizam duas ferramentas: uma chamada

tangle que permite a extracção automática de código-fonte compreensível por computadores, e a outra chamada *weave* que produz documentos legíveis e compreensíveis por pessoas. Esta técnica incentiva à escrita da documentação simultaneamente com a escrita de código, resultando potencialmente em programas de maior qualidade e de mais fácil manutenção. A Figura 2.1 ilustra o processo da técnica Literate Programming e os respectivos mecanismos de *tangling* e *weaving* de documentos.

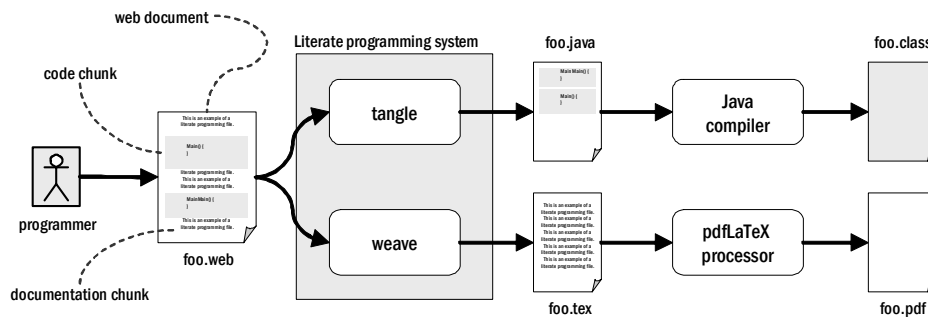


Figura 2.1: Literate Programming: *tangling* e *weaving* de documentos (Aguiar, 2003)

Apesar da elegância e das vantagens que oferece, esta técnica nunca foi utilizada em larga escala (fora dos ambientes académicos que a envolvem) devido às dificuldades de integração das ferramentas de Literate Programming nos ambientes de desenvolvimento existentes. Ademar Aguiar (Aguiar, 2003) sugere alguns motivos para a não adopção do Literate Programming em grande escala:

- a técnica requer a combinação de três linguagens: a linguagem de programação, a linguagem de formatação e a linguagem de interligação que permite a definição, referencia e inclusão dos pedaços de código nos documentos. Na versão original de Knuth estas linguagens eram Pascal, T_EX e WEB respectivamente;
- a técnica introduz demasiado *overhead* para projectos de pequena dimensão;
- o formato dos documentos Literate Programming são complexos, o que compromete seriamente a sua leitura e compreensão durante o processo de desenvolvimento;
- a grande parte das ferramentas de Literate Programming não é facilmente adaptável a projectos já existentes;
- a organização do código-fonte visto pelo compilador é diferente da organização original escrita pelo programador, o que geralmente causa problemas na utilização de ferramentas que façam manipulação de código-fonte, tais como, *debuggers*, geradores de código, ferramentas de *refactoring*, etc.

Evoluções do Literate Programming

Trabalhos recentes (Palmer and Hillenbrand, 2009) propõem solucionar alguns dos problemas identificados no Literate Programming original, nomeadamente a complexidade inerente à utilização das três linguagens distintas, com base numa nova linguagem – Ginger – especificamente desenhada para suportar Literate Programming.

Ginger utiliza G-expressions (Palmer, 2009) para representar código, documentação e ligações Literate. Como resultado, código e documentação são representados interna e externamente precisamente da mesma forma, proporcionando, assim, uma interface uniforme para pedaços de código e de documentação, o que permite uma manipulação, transformação e inspeção entre si, de tal forma, que as fronteiras entre código e descrição cognitiva se tornam muito ténues.

Embora partilhe muitas semelhanças com outros sistemas de Literate Programming, esta solução permite a unificação da experiência de programação Literate através da utilização de uma única linguagem, a linguagem Ginger.

Programas Literate escritos em Ginger utilizam um único interpretador que constrói uma árvore baseada em G-expressions, que pode ser facilmente transformada por forma a gerar, tanto documentação legível por pessoas, como código executável por computadores. Isto tem como efeito, por um lado a simplificação da experiência de programação para as pessoas, e por outro a possibilidade de melhoria na inspeção e manipulação de documentação e código.

As técnicas de documentação alternativas ao Literate Programming podem ser classificadas como métodos *single-source* e métodos *multiple-source*.

2.1.2 Métodos *single-source*

Os métodos *single-source* integram código-fonte e documentação num mesmo ficheiro, geralmente em ficheiros de código-fonte, nos quais a documentação é escrita na forma de comentários.

Como exemplos de métodos *single-source* vastamente utilizados é possível enumerar:

Javadoc (Sun Microsystems, 2003; Kramer, 1999) é uma ferramenta da Sun Microsystems para a geração de documentação da API da linguagem Java em formato HTML a partir de comentários de documentação no código-fonte.

Doxygen (van Heesch, 2002) é um sistema de documentação para diversas linguagens, nomeadamente C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, C#. O Doxygen permite a geração de documentação em diversos formatos, tais como, entre outros, HTML para geração de documentação *on-line*, e $\text{\LaTeX}$ para produção de manuais *off-line*.

Doc++ (Wunderling and Zockler, 2002) é um sistema de documentação similar ao Doxygen para as linguagens C, C++, IDL e Java e permite a exportação de documentação nos formatos HTML e $\text{\TeX}$.

Rdoc (RDoc, 2009) é um sistema de documentação para a linguagem Ruby que permite gerar documentação em formato HTML a partir de código-fonte Ruby com comentários.

PDoc (Langel, 2009) é um sistema de documentação escrito em Ruby para a geração de documentação da framework Script Prototype (Prototype, 2009) e das bibliotecas que utilizem esta framework.

Existem no mercado sistemas de documentação *single-source* para praticamente todas as linguagens, o que denota a importância deste método de documentação, principalmente para a documentação de APIs.

A documentação segundo os métodos *single-source* é extraída directamente a partir das fontes, o que torna muito mais fácil de manter a documentação em conformidade com o código-fonte. No entanto, a documentação produzida segundo estes métodos destina-se a documentação de baixo nível, geralmente para a produção de documentação de APIs, o que torna o método inadequado para a documentação de alto nível (requisitos, design, arquitectura, manutenção), na qual são necessárias referências transversais a diversos artefactos de software, tais como código-fonte ou diagramas UML.

2.1.3 Métodos *multiple-source*

Os métodos *multiple-source* integram conteúdos provenientes de diferentes fontes (documentos, código-fonte, modelos, etc), permitindo assim uma organização livre desses conteúdos, o que facilita o processo de compreensão a alto nível de sistemas. A documentação produzida segundo estes métodos é geralmente documentação externa, transversal aos sistemas.

No entanto, a garantia da consistência semântica (sincronismo) entre os artefactos utilizados na produção da documentação é um problema que geralmente estes métodos têm dificuldade em resolver.

A documentação externa tradicional, exemplo de um método *multiple-source*, é feita com recurso a editores de documentos (MS Word, $\text{\LaTeX}$, etc.), que permitem a produção de documentação utilizando livremente artefactos de software que são inseridos nos documentos por *copy-paste*. Este procedimento tem como principal vantagem a flexibilidade na produção de documentação de alto nível permitindo a livre integração de artefactos na documentação produzida. No entanto, tem como desvantagem a eventual perda de consistência entre os artefactos utilizados, uma vez que estes são copiados para os documentos, ficando inconsistentes à medida que vão sendo alterados ao longo do ciclo de vida normal do desenvolvimento de software.

Elucidative Programming

Alguns sistemas simulam a verosimilhança característica do Literate Programming (código-fonte e documentação escritos num mesmo documento) através de ferramentas que fazem a gestão automática das relações estreitas entre código-fonte e documentação. Um exemplo de um sistema deste género é a ferramenta de Elucidative Programming, o chamado *elucidator*, desenvolvida por Nørmark (Nørmark, 2000a), que permite a ligação entre o código fonte e documentos através da inserção de directivas especiais em ambos os lados. Uma característica especial dos *elucidators* é

a utilização de duas janelas sincronizadas dispostas lado a lado, para apresentar simultaneamente as duas partes da documentação.

Os *elucidators* inicialmente desenvolvidos para as linguagens Scheme e Java (Nørmark, 2000b; Nørmark et al., 2000) usam aplicações separadas para edição (Emacs) e navegação na documentação (*browser*). Um outro protótipo (Vestdam, 2003) integra um elucidador no Borland Together IDE (Borland, 2003) permitindo assim que ambas as actividades dentro do IDE.

Os *elucidators* suportam a definição das relações entre código e documentos com base no conhecimento extraído dos documentos e código-fonte, que é armazenada em estruturas de dados, no caso do Scheme Elucidator (Nørmark, 2000b), ou em bases de dados relacionais, no caso do Java Elucidator (Nørmark et al., 2000).

À semelhança do Literate Programming, a Elucidative Programming é destinado principalmente para documentação interna (Nørmark, 2000a), embora suporte também documentos externos estruturados, utilizando para tal, uma linguagem de marcação simples e dedicada.

Embora os *elucidators* não ofereçam o mesmo nível de incentivos à documentação da compreensão de programas oferecidos pelo Literate Programming, uma vez que o código e os documentos são editados e visualizados em janelas separadas, oferecem a vantagem de não exigir o processo específico de *tangling*, permitindo assim a edição e compilação directa do código, o que melhora a sua utilização no mundo real e a sua integrabilidade num universo mais alargado.

Documentação baseada em wikis

A utilização de wikis no desenvolvimento de software remota já à sua primeira utilização em 1995 quando Ward Cunningham criou o primeiro wiki, WikiWikiWeb (Cunningham, 1999), com o objectivo específico de criar um ambiente que permitisse interligar as experiências de vários programadores na descoberta de padrões de desenho de software e descreveu-o como “a mais simples base de dados *online* que possa eventualmente funcionar: *the simplest online database that could possibly work*”.

Wiki significa “rápido” em havaiano. Cunningham escolheu o nome WikiWikiWeb para o seu sistema porque achou mais interessante do que lhe chamar apenas “Quick Web”. Para além disso, em havaiano, as palavras são duplicadas para dar ênfase ao seu significado, e Cunningham queria deixar claro que o seu sistema era muito rápido.

A documentação em suporte web tem vindo a ser largamente adoptada devido às grandes vantagens que oferece, nomeadamente: baixo custo de publicação, acessibilidade, informação actualizada, facilidades de pesquisa e navegação, etc. (Aguiar and David, 2005), sendo, a documentação interna de interfaces de programação (APIs), como a do Java (Java 6 API, 2009), Ruby on Rails (Rails API, 2009), bons exemplos disso.

A produção de documentos na web requer, no entanto, ferramentas de edição capazes de suportar a estruturação e exploração de documentos hipertexto que não existem nos *web browsers* (Vitali and Bieber, 1999).

Os wikis utilizam linguagens simples de formatação de texto e um poderoso mecanismo de ligação dinâmica (Bodner and Chignell, 1999; Bodner et al., 1999) entre páginas, no qual os *links*

não são definidos estaticamente, mas sim calculados *on the fly* quando a página é carregada, com base na informação contextual, suportando assim a noção de páginas web adaptativas.

“Os wikis são abertos, permitem uma evolução incremental e orgânica, são de fácil edição e organização, promovem a convergência de conteúdos e termos, são tolerantes e de fácil acesso a outros utilizadores.” (Aguiar and David, 2005)

Os wikis oferecem um ambiente de autoria colaborativa de documentação baseada na web, o que, aliado ao facto de grande parte da documentação ser produzida actualmente em suporte web, faz com que os wikis possam ser, no mínimo, utilizados como uma boa ferramenta de edição, organização e armazenamento de documentação de software (Aguiar, 2003; Aguiar and David, 2005; Merson and Bachmann, 2005). Os wikis constituem assim uma boa ferramenta para documentação segundo os métodos *multiple-source* em formato web.

Dada a sua simplicidade e boa aceitação, em especial pela comunidade de software, os wikis estão hoje vastamente disponíveis e fortemente disseminados em diferentes domínios. Das várias implementações de wikis actualmente existentes, algumas disponibilizam funcionalidades específicas para suporte a desenvolvimento de software, como revisão de código, planeamento de projectos, engenharia de requisitos, rastreio de erros, testes, formatação de código-fonte, etc. O Trac (Software, 2009) é um sistema de *issue tracking* integrado com um wiki avançado para gestão de desenvolvimento de software, que permite a criação de links e referências entre *bugs*, tarefas, alterações, ficheiros e páginas wiki. O TWiki (Thoeny, 2009) disponibiliza plugins para gestão de projectos de eXtreme Programming. O SnipSnap (Jugel and Schmidt, 2003) permite a edição colaborativa de diagramas UML com base num sintaxe dedicada. Outros wikis, também relevantes para suporte ao desenvolvimento de software, são por exemplo o VeryQuickWiki (Cronin and Barnett, 2006) (base do XSDoc), o e o MoinMoin (Hermann and Waldmann, 2009) que faz, por exemplo, a coloração sintáctica de código-fonte de forma automática.

XSDoc O XSDoc – eXtensible Software Documentation – (Aguiar, 2003; Aguiar et al., 2003) é uma ferramenta de documentação aberta e extensível baseada num wiki e em tecnologia XML (Bray et al., 1998) que garante, seguindo uma abordagem *multiple-source*, a consistência semântica entre os diferentes artefactos utilizados na documentação de software.

O XSDoc foi criado para ultrapassar as limitações inerentes ao sistema de documentação Literate Programming (Knuth, 1984) e às técnicas alternativas, como o Elucidative Programming (Nørmark, 2000a), utilizadas para resolver o problema da inconsistência semântica característica da documentação de software, nomeadamente: interoperabilidade entre ferramentas, extensibilidade e integrabilidade de conteúdos heterogéneos. (Aguiar, 2003)

O XSDoc tem como intenção o fecho do fosso normalmente existente entre desenvolvimento e documentação de software, proporcionando um ambiente de documentação atractivo e eficaz, ao permitir a escrita de documentação em simultâneo com a escrita de código.

Arquitetura do XSDoc A infra-estrutura XSDoc é composta por um sistema wiki, *plugins* para integração em ambientes integrados de desenvolvimento, um conjunto de *templates* de documentos (*cookbooks*, *design patterns*, *overviews*, etc.), linguagens de anotação e conversores de conteúdos de e para o formato XML.

O componente principal do XSDoc é o XSDocWiki, um sistema wiki que estende um sistema wiki convencional, o VeryQuickWiki (Cronin and Barnett, 2006), com uma série de funcionalidades úteis para a documentação de frameworks e software em geral.

A documentação criada internamente no XSDocWiki pode integrar conteúdos externos, como código-fonte ou modelos, que requerem normalmente editores externos. Quando integrado num ambiente de desenvolvimento integrado através de *plugins* específicos, o XSDoc permite a autoria de todos os artefactos constituintes de um projecto de software, incluindo a autoria de documentação, num ambiente único, eliminando a necessidade da troca constante entre ferramentas durante o processo de desenvolvimento/documentação, disponibilizando inclusivamente, um *plugin* para o ambiente de desenvolvimento integrado Eclipse IDE (Foundation, 2009b).

Os documentos criados pelo XSDoc são codificados em formato XML e persistidos num repositório de conteúdos. Quando é feito o pedido de apresentação de um documento, o sistema extrai o documento armazenado no repositório em formato XML e processa-o, possibilitando a integração de conteúdos externos no documento, para a sua conversão no formato pedido. Para tal, conteúdos como código-fonte e diagramas UML, necessitam de um processamento especial porque, para serem integrados no documento, têm que ser convertidos do seu formato original para a sua representação no formato XML. Para tal, o sistema utiliza representações de conteúdos em dialectos como o JavaML 2 (Aguilar et al., 2004), Doxygen (van Heesch, 2002) e SVG/XMI (IBM AlphaWorks, 1999; Ferraiolo et al., 2003).

Mecanismo de integração Contrariamente às soluções de Literate Programming, que necessitam de uma linguagem de integração específica, para integração de documentação e código-fonte, o XSDoc utiliza como linguagem de integração entre documentação e conteúdos externos a própria linguagem de *markup* do wiki, que é bastante simples e de fácil aprendizagem. O XSDoc disponibiliza, para além das capacidades de hiperligação do próprio wiki, dois mecanismos dinâmicos de integração e sincronização de conteúdos externos: *inlining* de conteúdos, que permite apresentar conteúdos externos incluídos nas páginas do wiki, e *linking* de conteúdos que permite fazer hiperligações de páginas de documentação wiki para conteúdos externos como código-fonte e diagramas UML.

Os conteúdos estão sempre disponíveis para apresentação online através do XSDocWiki, mas podem também ser exportados para o formato HTML estático, para consulta *offline*, ou para o formato PDF, para impressão.

Os artefactos de código-fonte na linguagem Java utilizados no XSDoc são enriquecidos com formatação sintáctica e com hiperligações para outras partes do código, para outros ficheiros de código, para documentação Javadoc, ou outros documentos relacionados. Para tal, o sistema utiliza representações em XML do código-fonte – JavaML 2 – para pré-processamento.

Galaxy Wiki O Galaxy Wiki (Xiao et al., 2007) é uma proposta para um ambiente de desenvolvimento colaborativo baseado num wiki que conjuga as características dos wikis: edição colaborativa, aberto, de fácil acesso, links dinâmicos, etc., com funcionalidades de desenvolvimento, como edição, compilação, execução e *debugging* de código-fonte.

Este trabalho teve como base o reconhecimento dos wikis como sistemas ideais para documentação de software e o conceito de sincronismo entre artefactos desenvolvido para o XSDoc (Aguiar et al., 2003), que permite a inclusão dinâmica de artefactos como o código-fonte em páginas wiki. No entanto, a livre criação, edição e remoção de tais artefactos não é possível no próprio wiki uma vez que estes residem normalmente em sistema de controlo de versões externos ao wiki.

O suporte do Literate Programming numa base wiki foi experimentado no Galaxy Wiki com o objectivo de solucionar este problema. O Galaxy Wiki permite inserir código-fonte, utilizando, para tal, directivas dedicadas, por forma a permitir, num documento Literate, a sua visualização para documentação, assim como a compilação, execução e *debugging*.

O Galaxy Wiki foi construído com uma extensão ao MoinMoin (Hermann and Waldmann, 2009) e suporta as funcionalidades de criação, navegação, edição, remoção, compilação, execução e *debug* de programas Java.

Este trabalho lança um novo conceito de ambiente de desenvolvimento, ao qual chamaram de “engenharia de software orientado às páginas”. Este conceito parte da premissa de programar num wiki que contém uma colecção ilimitada de código-fonte e documentos interligados, facilmente editáveis através de *web browsers*.

Mozilla Developer Center (MDC) O projecto Mozilla é um projecto à escala mundial, patrocinado pela Mozilla Foundation (Mozilla Foundation, 2009), cujo principal objectivo é promover uma Internet aberta e participativa, através do desenvolvimento, utilização e promoção de normas abertas. Um componente fundamental do projecto é o desenvolvimento de aplicações *open-source*, como o *browser* Firefox e o cliente de email Thunderbird.

A fim de coordenar um projecto de software *open-source* de tão grande dimensão, é fundamental manter uma documentação de desenvolvimento actualizada sem grande esforço. A gestão deste projecto de documentação maciça é da competência do Mozilla Developer Center (Mozilla Developer Center, 2009). O Mozilla Developer Center fornece documentação sobre uma ampla gama de temas de desenvolvimento, incluindo a forma de desenvolver sites utilizando as mais recentes tecnologias web, programação em JavaScript, como compilar o Firefox e como contribuir para o projecto Mozilla.

Eric Shepherd relata (Shepherd, 2008) a evolução do sistema de suporte de documentação do Mozilla Developer Center desde um sistema inicial baseado no sistema de controlo de versões CVS, até à actual utilização de wikis.

A participação da comunidade num projecto deste género é fundamental e qualquer pessoa deve ser capaz de contribuir facilmente para a documentação. Para maximizar o número de participantes na documentação do projecto é importante não oferecer grandes entreves à sua entrada, facilitando o acesso e edição da documentação. No entanto, é necessário criar algum equilíbrio

com a necessidade de segurança para que não sejam adicionados conteúdos impróprios sem que tal seja notado.

O Mozilla Developer Center é um caso interessante porque tem mais de 40.000 utilizadores registados, e mais de 650.000 *page views* por dia (Shepherd, 2008).

Antes de mudar para uma solução baseada em wikis, a documentação da Mozilla Developer Center foi mantida numa versão HTML controlada usando o sistema de controlo de versões CVS. Esta solução exigia conhecimentos de HTML e não permitia visualizar a evolução dos documentos sem recurso ao CVS. O resultado foi um projecto de documentação relativamente estagnada, com contribuições oferecidas principalmente pelos engenheiros de software e raramente actualizado após a redacção inicial. Tornou-se claro que este era um problema grave, e foi decidido que um wiki seria uma melhor solução para a documentação do Mozilla Developer Center. A primeira opção foi a adopção do MediaWiki (Foundation, 2009c) devido à sua popularidade e maturidade. A partir de 2007, o Mozilla Developer Center tem vindo a ser suportado pelo MindTouch Deki (MindTouch Deki, 2009), uma plataforma colaborativa *open-source* baseada no conceito wiki.

Eric Shepherd aponta as seguintes *vantagens* na utilização de wikis: pelo facto de qualquer utilizador que lê a documentação poder contribuir, somos capazes de encorajar os programadores, utilizadores e leitores casuais a contribuir. Os programadores são incentivados a escrever artigos sobre temas que lhes são familiares. Outros utilizadores, mesmo sem grandes conhecimentos, poderão iniciar documentos de interesse, que estes serão certamente rapidamente evoluídos por quem mais domina o assunto. Da mesma forma, utilizadores sem grandes capacidades técnicas, como leitores casuais, poderão ajudar a identificar e corrigir erros factuais, mas mais frequentemente, encontram e corrigem erros de sintaxe e gramaticais. Os leitores descobrem ainda frequentemente falhas na documentação, importantes para a compreensão de um determinado tópico, podendo solicitar a documentação em falta. A utilização de wikis facilitou também a internacionalização dos conteúdos, em que voluntários se ocupam da produção do conteúdo traduzido.

Como *desvantagens* na utilização de wikis, Eric Shepherd aponta principalmente o risco de sabotagem e *spam*, que são problemas frequentes em sistemas web publicamente editáveis, nos quais se incluem os wikis, e a necessidade de monitorização constante e cuidadosa deste tipo de ocorrências.

O Mozilla Developer Center é um bom caso de estudo que revela a tendência para a adopção de wikis como sistemas de suporte à documentação de projectos de software, neste caso à escala mundial e num contexto *open-source*.

Dadas as suas características de sistemas abertos, orgânicos, de fácil edição e organização e a vasta gama de wikis disponíveis com funcionalidades específicas para suporte ao desenvolvimento de software, os wikis têm vindo a ser explorados como base de investigação na área da engenharia de software, e em particular na área da documentação. O XSDoc é um bom exemplo disso, ao expandir uma base wiki com algumas funcionalidades úteis para o desenvolvimento de software num poderoso sistema de documentação. Da mesma forma, o Galaxy Wiki suporta uma forma de Literate Programming num wiki com base numa extensão ao conhecido wiki MoinMoin.

2.2 Metodologias ágeis de desenvolvimento de software

Com o objectivo de melhorar a adaptação à constante mudança das tecnologias e requisitos, actualmente habituais nos projectos de desenvolvimento de software, começaram a surgir, no final da década de 90, novas metodologias que, embora com ideias diferentes, todas vieram dar uma maior ênfase a aspectos até então menos considerados, tais como:

- a estreita colaboração entre a equipa de desenvolvimento e o cliente;
- a comunicação face-a-face como forma de disseminação de conhecimento, tida como mais eficaz do que a escrita de documentação;
- a entrega frequente de novas funcionalidades como medida de valor para o cliente;
- a organização em pequenas equipas de desenvolvimento;
- formas de tolerar a rápida alteração dos requisitos e tecnologias ao longo do tempo.

Ainda sem o conceito actual de “ágil”, estas metodologias geraram bastante interesse em ambientes de desenvolvimento empresariais e académicos. A introdução da metodologia de desenvolvimento Extreme Programming (XP) (Beck, 2000) foi reconhecida como um ponto de partida importante para as várias outras abordagens de desenvolvimento ágil de software já existentes. Desde então, várias outras metodologias foram inventadas ou redescobertas como pertencentes à mesma família, tais como:

- Adaptive Software Development (Highsmith, 2000);
- Agile Modeling (Ambler, 2003b);
- Crystal Methods (Cockburn, 2004);
- Dynamic Systems Development Method (Stapleton, 1999);
- Feature Driven Development (S.R. and J.M., 2002);
- Lean Development (M. and T., 2003);
- Scrum (Schwaber, 1995; Schwaber and Beedle, 2002).

Em 2001, diversos membros da comunidade académica reuniram-se para discutir e delinear os *valores* que iriam permitir o desenvolvimento de software rápido e tolerável a mudanças. O grupo auto-denominou-se de “Aliança Ágil” (Agile Alliance, 2001), e o resultado do encontro de dois dias foi o manifesto denominado de “Manifesto para o Desenvolvimento Ágil de Software” (Agile Alliance, 2001). Nos meses que se seguiram, o grupo continuou o trabalho para criar os *princípios* da agilidade.

Importante	Mais importante
processos e ferramentas	pessoas e interações
documentação detalhada	software funcional
negociações contratuais	colaboração com o cliente
seguimento de um plano	resposta à mudança

Figura 2.2: Valores do manifesto da Aliança Ágil (Agile Alliance, 2001)

2.2.1 Valores

A Figura 2.2 sintetiza os valores nos quais assenta o manifesto. Embora a Aliança reconheça valor nos itens da coluna da esquerda, valoriza mais os itens da coluna da direita.

Pessoas e interações *versus* processos e ferramentas. A Aliança, muito embora reconheça importância nos processos institucionalizados e ferramentas de desenvolvimento, dá maior ênfase à relação e partilha no seio da equipa de desenvolvimento. Isso geralmente manifesta-se, nas metodologias existentes, em pequenas equipas de desenvolvimento distribuídas em ambientes que favoreçam a comunicação e relações entre os seus elementos constituintes, bem como na distribuição da responsabilidade sobre um projecto por todos os elementos da equipa.

Software funcional *versus* documentação detalhada. A documentação da compreensão de software não é vista como prejudicial mas o trabalho deve ser focado primordialmente no produto final, ou seja, na entrega frequente de software funcional. Para tal, o código deve ser mantido o mais simples e objectivo possível – “*do the simplest thing that could possibly work*” (Beck, 2000) – por forma a ser auto descritivo, fácil de compreender e de usar, deve ser apoiado em testes unitários, o que ajuda também na compreensão do design e da intenção da solução implementada. Assim, cabe à equipa determinar, para cada projecto em particular, a documentação que será absolutamente necessária. O software funcional é usado como medida principal de progresso no desenvolvimento, sendo prioridade máxima a satisfação do cliente através da entrega atempada e contínua de software usável.

Colaboração com o cliente *versus* negociações contratuais. A relação e colaboração entre a equipa de desenvolvimento e o cliente é tida como preferencial relativamente a contratos rígidos e pré-estabelecidos, muito embora seja dada importância a contratos bem definidos de acordo com a dimensão dos projectos. O processo de negociação deve ser visto como forma de garantir e manter uma relação viável com o cliente e não como definição rígida de todo o processo.

Resposta à mudança *versus* o seguimento de um plano. O grupo de desenvolvimento, constituído pela equipa de desenvolvimento e pelo representante do cliente, devem ter autonomia

suficiente para considerar eventuais ajustes que possam surgir durante o processo de desenvolvimento. Assim, a resposta à mudança é bem tolerada, mesmo em fases tardias do desenvolvimento.

2.2.2 Princípios

Com base nestes valores, a Aliança definiu os seguintes princípios:

1. Ter como prioridade máxima a satisfação do cliente entregando-lhe atempadamente e constantemente software funcional;
2. As alterações aos requisitos são bem vindas, mesmo em fases avançadas do desenvolvimento;
3. Entregas frequentes de software funcional, com intervalos de tempo que podem ser de algumas semanas a alguns meses, mas com preferência para intervalos mais curtos;
4. Representantes do cliente e programadores devem cooperar diariamente durante o desenvolvimento do projecto;
5. Construir projectos em torno de pessoas motivadas. Oferecer-lhes o ambiente e o suporte de que necessitam e confiar neles para que façam o seu trabalho;
6. O método mais eficaz de disseminar informação dentro de uma equipa de desenvolvimento é a conversação face-a-face;
7. O software funcional é a principal medida de progresso;
8. Manter um ritmo constante de trabalho, aplicável a todos os *stakeholders*, baseado num desenvolvimento sustentável;
9. Uma continua atenção para a excelência técnica e bom design melhora a agilidade;
10. A simplicidade, como arte de minimizar a quantidade de trabalho desnecessário, é fundamental;
11. As melhores arquitecturas, requisitos e design surgem de equipas auto-organizadas;
12. As equipas reflectem regularmente sobre a forma como se podem tornar mais eficazes, ajustando, em conformidade, o seu comportamento.

2.3 Documentação de software e agilidade

As actividades de comunicação e documentação num projecto de software começam normalmente bem antes do desenvolvimento. É usual cobrirem os processos, assim como o produto em

desenvolvimento, sendo produzidos normalmente diversos tipos de documentos, desde planos e calendários, a documentação de sistema, até à documentação para o utilizador.

A documentação de software envolve diversas questões, técnicas e não técnicas, das quais se enumeram de forma genérica as seguintes: (Aguiar, 2003)

- questões fundamentais de documentação técnica, tais como avaliação de qualidade;
- questões comuns da documentação de software relacionadas com os documentos em si;
- o processo de documentação adoptado;
- ferramentas de suporte à documentação.

Adicionalmente, podem surgir diversas questões específicas de um projecto em particular, tais como aquelas relacionadas com a natureza do produto (e.g. produtos reutilizáveis requerem mais esforço na documentação), ou os requisitos do domínio específico, relacionados com normalizações, certificações ou legislação.

Para ser eficaz, a documentação de software disponibiliza normalmente *múltiplas vistas* sobre um sistema (estática, dinâmica, interna, externa) a *diferentes níveis de abstracção* (arquitectura, design, implementação). Mediante o aspecto concreto a documentar, poderá ser conveniente utilizar e combinar conteúdos representados em *diferentes notações* (texto, código, modelos, imagens) provenientes de *diferentes tipos de artefactos* (documentos de requisitos, especificações de casos de utilização, *drafts* de design, manuais de referência, ficheiros de código-fonte e modelos) produzidos em diferentes alturas do ciclo de desenvolvimento, de forma colaborativa e por diferentes tipos de pessoas (Aguiar and David, 2005).

A quantidade de esforço alocado à documentação varia de projecto para projecto, mas o tipo de processo adoptado tem uma importante influência. Os processos tradicionais de desenvolvimento usam normalmente orientações normativas, demasiada formalidade, e grandes quantidades de documentação com o intuito de garantir os níveis necessários de profissionalismo, disciplina e comunicação num projecto. Os processos ágeis partilham os mesmos objectivos, mas pretendem atingi-los valorizando mais outros aspectos de um projecto de software, nomeadamente a comunicação, a simplicidade, o feedback e a coragem, tal como já descrito na secção anterior.

Os processos ágeis de desenvolvimento de software (Agile Alliance, 2001) são muito restritivos relativamente à documentação sugerindo que: “A documentação deve concisa e objectiva” (Ambler, 2003a).

Scott Ambler, no seu trabalho sobre “Modelação Ágil” (Ambler, 2003b) focou-se essencialmente nos princípios e práticas para uma modelação eficaz, mas rapidamente descobriu que tal não seria suficiente, e que deveria considerar também o problema da criação e manutenção de documentação, o que deu origem ao seu trabalho sobre “Documentação Ágil” (Ambler, 2003a), do qual são aqui apresentados alguns dos principais tópicos.

2.3.1 Modelos, documentos e código-fonte

A Figura 2.3 descreve a relação entre modelos, documentos, código-fonte e documentação. Do ponto de vista da “Modelação Ágil” um documento é qualquer artefacto externo ao código-fonte, cujo objectivo seja transmitir informação de forma persistente. O conceito de um documento é diferente do conceito de um modelo, que é uma abstracção para descrever um ou mais aspectos de um problema ou uma potencial solução para resolver um problema. Alguns modelos podem vir a tornar-se documentos, ou ser incluídos como parte destes, embora muitos deles serão simplesmente descartados depois de cumprida a sua finalidade. Alguns modelos serão utilizados para suportar o desenvolvimento do código-fonte, embora alguns deles possam apenas ser utilizados para orientar o desenvolvimento de outros modelos. Embora o código-fonte, como uma sequência de instruções, incluindo os comentários que descrevem as instruções, para um sistema de computador, seja claramente uma abstracção, ainda que detalhada, no âmbito da “Modelação Ágil”, não será considerado um modelo. Além disso, para fins de discussão o termo documentação inclui tanto os documentos, como os comentários no código-fonte (Ambler, 2003a).

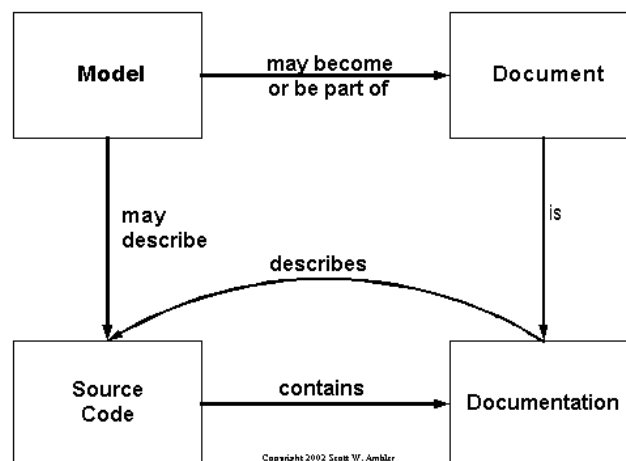


Figura 2.3: Relação entre modelos, documentos, código-fonte e documentação (Ambler, 2003a)

2.3.2 Circunstâncias que justificam a documentação

Scott Ambler sugere que a documentação deverá ser produzida apenas nas seguintes circunstâncias (Ambler, 2003a):

Quando é um requisito definido pelos *stakeholders* de um projecto A criação de documentação é fundamentalmente uma decisão de negócio. Os *stakeholders* de um projecto deverão estar cientes dos custos e vantagens da produção e manutenção de documentação e devem ter uma decisão final quanto a esse investimento.

Para definir um modelo contratual Modelos contratuais definem a forma como um sistema interage (unidireccional, ou bidireccionalmente) com um outro sistema externo. Estes modelos

são muitas vezes necessários quando um grupo externo controla um recurso de informação que o sistema exige, tal como uma base de dados, uma aplicação legada ou um serviço de informação. A prática de “Formalizar modelos contratuais” da “Modelação Ágil” (Ambler, 2003b) estipula que um modelo contratual é algo que ambas as partes deverão acordar mutuamente para, documentar e manter ao longo do tempo, se necessário. É importante salientar que o desenvolvimento de um modelo contratual deve ser validado pelos *stakeholders* do projecto.

Para melhorar a comunicação com uma entidade externa Nem sempre é possível ter equipas de desenvolvimento e *stakeholders* reunidos num mesmo local a todos os momentos. Nestes casos, a documentação partilhada, pode ser combinada com mecanismos de comunicação alternativos, como a conversação face-a-face, email, chat, ou ferramentas colaborativas. A documentação não deve, no entanto, ser usada como o principal meio de comunicação uma vez que é susceptível a erros de interpretação, muito embora seja um bom mecanismo de apoio. A documentação deverá ser encarada, nesta situação, como um último recurso.

Para a manutenção da memória de um sistema Um dos princípios da “Modelação Ágil” de “Facilitar o próximo esforço é o seu segundo objectivo” pode ser entendido como um equilíbrio ao princípio de “Software funcional é o seu principal objectivo”. Desta forma, é importante ter presente que não deve ser alocado apenas esforço no desenvolvimento de software, mas também na produção de documentação de suporte necessária para a utilização, operação, suporte e manutenção do software ao longo do tempo.

Para fins de auditoria Certas organizações são obrigadas a apresentar provas de como cumprem requisitos ou seguem determinados processos definidos por entidades reguladoras. Para tal deverá ser produzida a documentação estritamente necessária.

Para melhor compreender um assunto O acto de escrever e colocar ideias sobre um sistema na forma de um documento, pode ajudar a solidificar conhecimentos e a descobrir potenciais problemas inicialmente ocultos.

2.3.3 Características de um documento ágil

Quanto a Scott Ambler, um documento pode ser considerado ágil quando satisfaz os seguintes critérios (Ambler, 2003a):

Documentos ágeis maximizam o retorno do investimento (ROI) dos *stakeholders* O benefício proporcionado por um documento ágil é maior do que o investimento na sua criação e manutenção e, idealmente, o investimento feito nessa documentação terá sido a melhor opção para alocar os recursos gastos.

Os custos de um documento são conhecidos pelos *stakeholders* Os *stakeholders* devem conhecer o custo (TCO) de um documento e devem estar dispostos a investir na sua criação e manutenção.

Documentos ágeis são “concisos e suficientes” Um documento ágil contém apenas a informação necessária para cumprir o seu objectivo, ou seja, é tão simples quanto possível. Por exemplo, partes de um documento podem ser escritas usando listas em vez de parágrafos. Desta forma, é possível passar a informação essencial sem a necessidade de investir tempo na composição e aspecto do texto, o que está alinhado com o princípio “O conteúdo é mais importante que a representação” (Ambler, 2003b). Os documentos ágeis contêm frequentemente referências para outras fontes de informação e devem seguir o princípio “Assumir a Simplicidade” (Ambler, 2003b) e seguir a prática “Criar conteúdos simples”(Ambler, 2003b) sempre que possível.

Documentos ágeis cumprem um objectivo Documentos ágeis são coesos e cumprem um determinado objectivo bem definido. Caso não estejam claros e bem identificados os motivos pelos quais está a ser elaborado um documento, ou o seu propósito seja duvidoso, a actividade de documentação deve ser interrompida e efectuada uma reflexão.

Documentos ágeis descrevem assuntos que realmente interessam Documentos ágeis focam informações críticas e pouco óbvias, como lógica de concepção, requisitos, procedimentos de uso, ou dos procedimentos operacionais.

Documentos ágeis têm um cliente específico e minimizam o esforço desse cliente A documentação de um sistema é normalmente escrita para responsáveis de manutenção, disponibilizando uma visão geral sobre a arquitectura do sistema e, possivelmente, requisitos críticos e decisões de design. A documentação de utilizador inclui já, frequentemente, tutoriais para a utilização de um sistema escrito numa linguagem compreensível pelos utilizadores, enquanto que a documentação de operações descreve como executar o sistema e é escrita numa linguagem que a equipa de operações possa compreender. Diferentes clientes requerem diferentes tipos de documentos e, muito provavelmente, diferentes estilos de escrita. A proximidade com o cliente é importante para a percepção do tipo de documentação que este necessita. De outra forma, incorre-se no risco de criar demasiada documentação, ou documentação desnecessária, perdendo-se assim agilidade.

Documentos ágeis são suficientemente consistentes, detalhados e precisos Um documento ágil não precisa de ser perfeito. Deverá apenas conter o suficiente para que os objectivos definidos sejam cumpridos.

2.3.4 Questões relacionadas com a documentação

Os Agilistas reconhecem que a produção de documentação eficaz é um compromisso (*trade-off*), em que o objectivo é obter a documentação suficiente, na altura certa, para um determinado

público alvo. Para que tal seja cumprido é necessário ter em conta uma série de questões que as equipas de projectos devem analisar, das quais Scott Ambler salienta as seguintes: (Ambler, 2003a)

Desenvolvimento de software versus desenvolvimento de documentação Estamos perante o compromisso fundamental que estará presente em qualquer projecto de software: todo o tempo gasto em documentação é tempo não utilizado no desenvolvimento de novas funcionalidades. Se num extremo temos projectos em que não é produzida qualquer documentação, no outro extremo estão os projectos em que não é produzido software. Nenhum dos extremos é o indicado. Se por um lado é importante produzir software funcional (princípio “Software funcional é o seu principal objectivo” (Ambler, 2003b)), por outro é importante viabilizar e facilitar o próximo desenvolvimento (princípio “Facilitar o próximo esforço é o seu segundo objectivo” (Ambler, 2003b)). A necessidade e o nível de documentação será sempre dependente do tipo de sistema a documentar, mas será sempre necessário escrever alguma documentação.

Especificações executáveis oferecem maior valor que documentação estática A maioria da informação capturada em documentos de especificações tradicionais, tais como especificações de requisitos, especificações de arquitectura, ou especificações de concepção, pode ser capturada como “especificações executáveis” em forma de testes. Uma abordagem de *Test-Driven Development* (TDD) obriga à escrita de especificações detalhadas de forma eficaz. Especificações executáveis, como testes de comportamento que especificam as necessidades do cliente, ou testes unitários de desenvolvimento que especificam detalhes de design, oferecem um valor significativo para os programadores porque não só ajudam a especificar o sistema, como também ajudam a validá-lo. Uma vez que estes artefactos têm valor para o sistema, é muito provável que venham a ser mantidos actualizados. Contudo, os testes não cobrem certamente todas as necessidades de documentação, sendo necessária a escrita de documentação estática, mas especificações executáveis devem ser desenvolvidas sempre que possível.

Distribuição de tarefas É importante definir quem deverá efectuar a documentação. Se por um lado o técnico de documentação tem as capacidades de documentar, por outro o programador conhece melhor o sistema a documentar. Algumas equipas optam por canalizar o trabalho para o técnico de documentação, retirando esforço ao programador mas introduzindo uma maior probabilidade de ocorrência de erros por desconhecimento do funcionamento do sistema. Uma outra abordagem, que poderá ser mais vantajosa, é incentivar o programador a produzir um documento base sobre o qual o técnico de documentação deverá trabalhar. Scott Ambler dá preferência a um terceiro cenário, em que o técnico de documentação e o programador trabalham em conjunto, aprendendo um com o outro.

Alteração dos requisitos de documentação ao longo do projecto As exigências de documentação durante o desenvolvimento são diferentes das exigências pós-desenvolvimento. Durante o desenvolvimento estão a ser exploradas questões de implementação como o que será necessário construir ou como interagem os componentes a desenvolver, enquanto que em pós-desenvolvimento é comum tentar compreender o que foi desenvolvido, porque foi desenvolvido de determinada forma e como funciona. Em geral é dada preferência a uma documentação menos exaustiva durante o desenvolvimento e mais pormenorizada e formal após o desenvolvimento.

Quando documentar Num extremo temos a escrita da documentação em paralelo com o desenvolvimento do software. Tem como vantagem o facto de ser mais fácil apreender informação relevante ao longo do desenvolvimento, mas tem como desvantagem a necessidade de reajuste da documentação sempre que é efectuado *refactoring* ao código. Esta prática, para além de atrasar o desenvolvimento, resulta num desperdício de esforço. Enquanto os requisitos não estabilizarem, a documentação excessiva poderá tornar-se muito dispendiosa manter. Outro extremo é documentar apenas depois de concluído o desenvolvimento, sendo que a principal vantagem é ter a garantia que se está a documentar algo estável. Existem, claramente, uma série de desvantagens nesta abordagem, como o esquecimento de determinados detalhes e decisões efectuadas ao longo do desenvolvimento, a possível falta de verbas, ou mesmo a falta de vontade em documentar. Um equilíbrio eficaz consiste em documentar assim que o projecto estabilize.

Documentar depois da informação estabilizar Se for efectuada documentação antes da informação estabilizar é muito provável que seja necessário reajustar a documentação de acordo com as alterações da informação. É recomendado aguardar que a informação estabilize e assim compreender que informação poderá ser verdadeiramente útil. Neste cenário o único problema é que o trabalho de documentação poderá ficar um pouco atrasado em relação ao trabalho de desenvolvimento.

Documentação no código versus documentação externa Onde colocar a documentação? No código-fonte, escrevendo código “auto-documentado”, ou colocando toda a documentação em artefactos externos? Mais uma vez é necessário encontrar um equilíbrio eficaz. Quando o público alvo são programadores, o melhor local para colocar a maior parte da documentação é o código-fonte, muito embora possa vir a ser necessária alguma documentação geral do sistema. No entanto, o público alvo da documentação de um sistema não são apenas programadores, e, como tal, poderão não entender, ou mesmo ter acesso ao código-fonte. Este tipo de público alvo necessita de documentação externa que responda precisamente às suas necessidades. A prática “Fonte única de informação” (Ambler, 2003b) sugere que seja feito um esforço por obter informação apenas uma vez, e no melhor local possível.

Quantidade versus qualidade A quantidade na documentação pode ser útil por conter provavelmente a maioria das informações necessárias para a manutenção e evolução de um sistema. No

entanto, uma grande quantidade de informação é susceptível de conter um número significativo de erros e é de difícil manutenção. Um documento mais curto, muito provavelmente não irá conter informações tão detalhadas, mas que poderá fornecer informação importante que conduzirá a código-fonte, ou outros documentos, para a obtenção de mais detalhes. Este tipo de documentos é geralmente mais fiável por ser mais curto e conter normalmente apenas conceitos de alto nível, como a arquitectura do sistema que muda, de forma geral, mais lentamente do que as minúcias detalhadas contidas num documento maior, podendo, portanto, ser mais facilmente actualizado ou simplesmente reescrito. Um documento maior não é necessariamente de menor qualidade do que um menor, mas é provável que seja percebido como tal, até prova em contrário.

2.3.5 Boas práticas na documentação de software

Embora o trabalho de Scott Ambler sobre “Modelação Ágil” (Ambler, 2003b) não seja centrado nos problemas da documentação, a problemática que aborda para a criação rentável de modelos foi considerada como a principal referência por Ademar Aguiar (Aguiar, 2003) para a problemática da produção rentável de documentação de frameworks. Aguiar (Aguiar, 2003) descreve uma combinação de “boas práticas” para a produção de documentação de frameworks, seguindo uma abordagem minimalista, que considera útil para a produção de documentação rentável, ou seja, cujos benefícios possam ser superiores aos custos de produção e manutenção. Estas práticas são a seguir apresentadas, e pressupõem, que o conteúdo dos vários documentos sejam mantidos em coerência com o apoio de um ambiente e ferramentas de documentação adequados. Sem esse apoio, os custos da adopção de algumas dessas práticas seria totalmente diferente, por exemplo, a prática de “Criar vários documentos de uma só vez”.

Propriedade colectiva Regra geral, todos os documentos são legíveis e editáveis por qualquer pessoa envolvida no projecto. A propriedade colectiva dos documentos leva geralmente a documentos de melhor qualidade, uma vez que todos podem contribuir, resultando em documentos mais ricos e completos. Os documentos poderão posteriormente ser revistos por um responsável técnico para melhorar a sua homogeneidade, consistência de termos, estilo de escrita e formatação.

Escrita colaborativa A escrita em colaboração com diferentes pessoas permite a avaliação da compreensão, perfeição e fidelidade do documento.

Criar documentos simples, mas simples o suficiente Um documento minimalista deve ser breve, não devendo conter tudo, mas apenas a informação suficiente para que cumpra a sua finalidade e apenas para o seu público-alvo. A simplicidade e compreensão dos conteúdos devem ser avaliados pelos leitores.

Criar vários documentos de uma vez Para representar todos os aspectos de uma framework, e servir todas as audiências e propósitos, é necessário o uso de diferentes documentos (e.g, receitas, exemplos, descrições de *hooks*, padrões), que quando editados em paralelo ajudam ao “despejo”

do conhecimento por parte dos autores de forma mais eficaz, uma vez que lhes é permitido documentar praticamente todos os aspectos que têm em mente, sem a necessidade de troca de contextos. Referências cruzadas devem ser usadas para interligar os documentos separados, mas relacionados.

Publicar documentos publicamente Documentos publicamente disponíveis, promovem a transferência de conhecimento, e melhoram a comunicação e a compreensão. O feedback dos leitores é beneficiado e a qualidade global dos documentos é rapidamente melhorada.

Documentar e actualizar apenas quando necessário Para ser rentável, os documentos devem ser criados e iterativamente refinados apenas quando necessário, e não quando desejado.

Reutilização de documentação A reutilização de conteúdos e estruturas de documentação existente, melhora a produtividade e a qualidade da documentação. A reutilização de conteúdos deve ser modular, fechada, e legível em qualquer ordem.

Usar ferramentas simples A utilização de ferramentas simples ajuda a focar mais nos conteúdos, em vez de dispersar a atenção na apresentação.

Definir e seguir normas de documentação Os autores devem concordar e seguir um conjunto comum de normas e convenções de documentação num projecto.

Documentar para compreender Documentar ajuda na formalização das ideias focadas em aspectos singulares, isolados de muitos outros.

2.3.6 Algumas práticas de Extreme Programming

Alguns exemplos de práticas do Extreme Programming (XP) (Beck, 2000) que ajudam a melhorar a comunicação entre a equipa sem necessidade de recurso a demasiada documentação são também descritas abaixo:

User stories É uma técnica simples de comunicar requisitos e de monitorizar o progresso de um projecto, fácil de adoptar, tanto por programadores como por clientes.

Testes de aceitação Ajudam na documentação dos requisitos do cliente de uma forma mensurável de testável.

Testes unitários Ajudam na comunicação do design de uma classe porque mostram exemplos concretos da forma de utilização da classe.

Presença do cliente no local de desenvolvimento Ajuda na comunicação entre programadores e o cliente, ou alguém que o represente, tipicamente um representante seu como por exemplo um gestor de produto.

Programação em pares Força a comunicação constante entre programadores, tendo como consequência uma aprendizagem mútua e contínua, o que leva a um equilíbrio da equipa.

Código simples Significa a escrita de código o mais simples possível que permita a resolução de um problema. Isto ajuda na comunicação entre código e programadores uma vez que código simples é mais fácil de compreender, tornando-se auto-documentado, evitando comentários de documentação e os problemas inerentes de estes virem a ficar desactualizados ao longo do tempo.

Refactoring contínuo Ajuda a manter o código e design o mais simples possível.

O código é pertença de todos Força revisões contínuas do código por todos os membros da equipa a todo o momento, o que ajuda na comunicação entre membros da equipa, e ajuda a evitar situações nas quais toda a equipa depende de uma só pessoa.

Quando efectivamente adoptadas, estas práticas contribuem para melhorar a comunicação do projecto (cliente & programadores, programador & programador, programador & código) e conduzir a código e design auto-documentados, mais simples de ler e compreender.

Este conjunto de boas práticas e recomendações constitui um “guião” que tem como objectivo principal aumentar a rentabilização do investimento do cliente em processos ágeis de desenvolvimento de software, através de um equilíbrio na produção de documentação. No entanto, a sua adopção requer algum cuidado, devendo ser feito um doseamento destas práticas na medida de cada projecto em particular.

Considerando a importância da documentação para o sucesso de projectos de software, Andreas Rueping fornece também recomendações de como produzir documentação ágil de software, recorrendo, para tal, a um formato de padrões (Rueping, 2003).

Capítulo 3

Problema

A documentação de software é uma tarefa árdua e geralmente evitada. Tal deve-se ao facto de:

1. ser uma tarefa não apreciada por parte de quem desenvolve;
2. o seu benefício não ser devidamente valorizado nem imediato;
3. não existirem ferramentas adequadas que facilitem e motivem a prática da documentação.

Um dos motivos apontados por Ademar Aguiar (Aguiar, 2003) para a não documentação é a falta de ferramentas adequadas que permitam criar facilmente uma representação rica da informação mental dos programadores na forma de documentos. O autor sugere que o processo de documentação deve ser o mais automatizado possível por forma a reduzir ao máximo o custo de produção sem prejuízo da qualidade da documentação produzida, devendo o processo de documentação ser flexível e facilmente adaptável a diferentes projectos e ambientes de desenvolvimento. O autor refere ainda exemplos de actividades de documentação que podem beneficiar do suporte de ferramentas adequadas:

Ferramentas de autoria Ferramentas atractivas, interoperáveis e de fácil utilização para auxiliar os editores na produção e manutenção de documentação poderia ser um importante incentivo para mais e melhor documentação. A interoperabilidade entre ferramentas de documentação com ambientes de desenvolvimento integrados é simultaneamente um problema técnico e de normalização.

Sincronização de conteúdos A utilização de ferramentas ou mecanismos automáticos para sincronizar código-fonte, modelos e documentos é crucial para a preservação da consistência entre todos os conteúdos, especialmente na presença de redundância. Tal sincronização permitiria manter especificações de design constantemente actualizadas, constituindo um importante incentivo à documentação na fase de desenvolvimento.

Ferramentas de extracção Com o objectivo de reduzir o esforço na produção e utilização de documentação, é, por vezes, desejável produzir, através de ferramentas de engenharia reversa,

vistas pré-definidas de artefactos de software, tais como, código-fonte, modelos e especificações formais. Exemplos comuns são a geração de modelos UML directamente do código-fonte e vice-versa, actualmente suportadas em várias ferramentas de desenvolvimento, como Together Control Center (ex-Borland) (Borland, 2003) e Eclipse IDE (Foundation, 2009b). Exemplos de automatismos mais sofisticados são a identificação e classificação de *hot-spots* de frameworks, ou a descoberta de padrões (ou meta-padrões) de software instanciados na implementação de frameworks (Flores and Aguiar, 2006).

Facilidades de navegação e pesquisa Dada a elevada quantidade de informação normalmente contida na documentação de projectos de grande dimensão ou complexidade, tais como frameworks, é importante oferecer ao utilizador facilidades de navegação e recuperação de informação, para que este não se perca na documentação. Tais facilidades incluem a selecção, filtragem, pesquisa, navegação e mecanismos de histórico.

3.1 Definição do problema

O problema identificado neste trabalho pode ser definido como a falta de ferramentas de apoio à documentação de software que possibilitem em simultâneo:

1. a documentação de software de alto nível (documentação externa);
2. a preservação da consistência semântica entre artefactos utilizados na documentação produzida;
3. a escrita colaborativa de documentação;
4. a publicação da documentação produzida em suporte web;
5. independência relativamente às ferramentas de autoria de documentação;
6. suporte para diferentes tipos de projectos, tanto ao nível da dimensão, como ao nível das tecnologias e linguagens utilizadas;
7. facilidade de adopção e utilização.

Este problema é focado para a realidade concreta dos projectos ágeis, especialmente para projectos *open-source*, nos quais a documentação é de extrema importância para a disseminação do conhecimento, e para a qual a flexibilidade de ferramentas de documentação é crucial para a sua adopção e utilização.

3.2 Justificação do problema

As técnicas de documentação existentes, descritas na Secção 2.1, resolvem pontualmente alguns dos itens delineados acima, mas nunca todos em simultâneo.

3.2.1 Literate Programming

A técnica de documentação proposta por Donald Knuth é a que mais incentiva a escrita de documentação em simultâneo com o desenvolvimento de software, pressupondo, aliás, que a escrita da documentação seja feita ainda antes da escrita do código, garantindo assim a persistência da informação mental do autor do código no documento.

A sincronização de conteúdos é naturalmente garantida, uma vez que a própria técnica assenta, em grande parte, neste princípio.

No entanto, a técnica de Literate Programming apresenta sérios problemas já apontados na secção 2.1.1, que impedem a sua adopção em grande escala. No contexto deste trabalho destacam-se os seguintes problemas:

Falta de interoperabilidade A técnica de Literate Programming requer ferramentas dedicadas para a sua utilização, o que compromete a sua interoperabilidade com outras ferramentas de desenvolvimento e documentação, bem como a independência relativamente às linguagens de programação suportadas.

Dificuldade de utilização A técnica de Literate Programming obriga à utilização constante e simultânea de três linguagens – linguagem de formatação (normalmente $\text{\LaTeX}$), linguagem de interligação e linguagem de programação – o que introduz uma complexidade acrescida dificultando todo o processo de desenvolvimento/documentação.

Estes dois problemas são suficientemente dissuasores à adopção desta técnica para produção de documentação ágil de software.

Evoluções do Literate Programming

Os trabalhos recentemente surgidos de novas formas de Literate Programming, nomeadamente o Galaxy Wiki, uma solução totalmente baseada em tecnologia wiki, e a recentemente apresentada solução baseada na linguagem Ginger vêm colmatar de certa forma estes problemas. No entanto, estas soluções estão ainda em fases prematuras de investigação, pelo que não oferecem a adaptabilidade pretendida para os diferentes tipos de projectos e linguagens suportáveis.

3.2.2 Métodos single-source

Os métodos *single-source*, como o Javadoc, Doxygen, etc. (Sun Microsystems, 2003; van Hesch, 2002), facilitam, mas não garantem, a manutenção da consistência semântica entre código, documentação e mesmo modelos, pela proximidade dos conteúdos, uma vez que toda a documentação é escrita no código-fonte em forma de comentários. Os comentários de documentação permitem formatação, possibilitando a produção de documentação rica, em alguns casos, com ligações directas a qualquer parte do código.

A escrita colaborativa da documentação é garantida da mesma forma que é garantida a escrita de código, geralmente pela utilização de sistemas de controlo de versões, uma vez que o código e documentação são escritos num mesmo ficheiro.

A produção de documentação em suporte web é naturalmente garantida por estes métodos, uma vez que se destinam geralmente à publicação neste tipo de suporte de documentação de APIs.

A adopção dos métodos *single-source* é relativamente fácil uma vez que existem ferramentas de documentação para praticamente todas as linguagens. A sua utilização é simples, limitando-se a comentários com alguma formatação, muito embora a formatação possa variar com a ferramenta e, consequentemente, com a linguagem a documentar.

Esta técnica de documentação resolve praticamente todos os problemas identificados neste trabalho, no entanto, a documentação produzida segundo estes métodos será sempre limitada à escala de propriedades, métodos, classes, ou packages, sendo desadequada para a produção de documentação transversal de alto nível.

3.2.3 Métodos *multiple-source*

Nos métodos *multiple-source*, do qual é exemplo a documentação convencional externa de software, a documentação é geralmente constituída por documentos de texto nos quais são inseridos conteúdos externos como código-fonte ou modelos provenientes do projecto de desenvolvimento, normalmente por *copy/paste*. A produção de documentação segundo estes métodos pode ser feita de forma colaborativa em ambientes web, utilizando ferramentas como wikis ou outras ferramentas colaborativas em suporte web, tais como o Google Docs (Google, 2009b). Muito embora estes métodos permitam a escrita de documentação de alto nível, sofrem geralmente do problema de falta de sincronismo entre os diversos artefactos utilizados, obrigando a um esforço significativo para manter a documentação actualizada.

Elucidative Programming

Elucidative Programming (Nørmark, 2000a) é uma variação interessante do Literate Programming original que permite a documentação da compreensão dos programas, apresentação online, e preservação do código fonte, a custo da separação dos ficheiros de código-fonte dos ficheiros de documentação, sendo a preservação da coerência semântica das relações entre o código fonte e documentos feita por mecanismos automáticos. Ao manter o código fonte (praticamente (Nørmark, 2000b)) intacto, e usando documentos XML, os *elucidators* (ou seja, as ferramentas de Elucidative Programming) integram bem com os editores de código-fonte e ferramentas prevenindo assim os problemas de integração dos sistemas tradicionais de Literate Programming.

No entanto, os documentos (eDocs) são limitados a estruturas simples com base em capítulos, secções e parágrafos. As relações entre o código-fonte e documentos são também limitadas permitindo apenas três tipos de relacionamentos: dentro de documentos, dentro do código, e entre o código fonte e documentação. Uma limitação adicional é a falta de suporte para a integração de conteúdos noutras representações, de entre as quais o UML é a mais importante.

Com a finalidade de potenciar o *elucidator* para Java, foram desenvolvidos trabalhos (Aguiar and Vestdam, 2001; Aguiar and Vestdam, 2002) com o objectivo de flexibilizar o modelo interno

do elucidador, para suportar a integração de novos tipos de conteúdos, assim como para a elaboração de um protótipo para integração do Java Elucidador na ferramenta Borland Together (Borland, 2003). O modelo interno foi, no entanto, considerado demasiado difícil de evoluir, tendo, a integração no Borland Together, sido concluída mais tarde por Vestdam (Vestdam, 2003).

Contudo, a utilização desta técnica fica algo limitada ao suporte existente no que diz respeito aos ambientes de desenvolvimento e linguagens suportadas. Adicionalmente, e tal como no caso do Literate Programming, a adopção do Elucidative Programming é demasiado trabalhosa para projectos já existentes uma vez que implica inserir comentários especiais no código para ligação à documentação.

Documentação baseada em wikis

Os wikis podem ser utilizados como ferramentas para documentação externa de software segundo os métodos *multiple-source* uma vez que permitem a criação de documentos, tal como qualquer outro editor convencional. Adicionalmente, a utilização de wikis para a produção de documentação de software permite a edição colaborativa, publicação natural em suporte web, facilidade de organização e crescimento orgânico da documentação, possibilidade de definição de templates para documentos tipo, entre outras vantagens enumeradas anteriormente na Secção 2.1.3.

Wikis como o MoinMoin e o SnipSnap, fazem inclusivamente a formatação automática de código-fonte para diversas linguagens, possibilitando uma melhor leitura do código-fonte utilizado na documentação.

No entanto, nenhum dos wikis actualmente disponíveis no mercado resolve o problema da falta de sincronismo entre artefactos produzidos, obrigando a colocar conteúdos externos por *copy/paste* nas páginas de documentação.

XSDoc A infra-estrutura XSDoc utiliza as potencialidades oferecidas pelos wikis para a produção de documentação, adicionando-lhe funcionalidades que possibilitam referenciar conteúdos externos para inclusão nos documentos, garantindo assim o sincronismo entre os artefactos utilizados na produção de documentação. Esta funcionalidade coloca o XSDoc um passo à frente relativamente aos wikis convencionais, ao resolver o principal problema identificado na utilização dos wikis para a produção de documentação de software.

O XSDoc oferece ainda a possibilidade de integração em ambientes de desenvolvimento integrado, o que potencia bastante a escrita da documentação em simultâneo com o desenvolvimento do projecto.

No entanto, a infra-estrutura XSDoc não resolve por completo o problema identificado neste trabalho, manifestando as seguintes limitações:

- é de difícil instalação e configuração uma vez que depende de várias tecnologias para processamento de conteúdos;
- depende do acesso local por sistema de ficheiros aos artefactos de software para as funcionalidades de sincronismo de conteúdos, o que torna a infra-estrutura pouco flexível e de difícil

adopção, uma vez que estes artefactos residem normalmente remotamente em sistemas de controlo de versões;

- oferece apenas suporte para artefactos de código-fonte na linguagem Java e C++;
- obriga à utilização do seu wiki base – XSDoc Wiki – para a produção de documentação, o que impede a utilização da infra-estrutura noutros suportes de autoria de documentação, tais como outros wikis.

Apesar dos problemas identificados, a infra-estrutura XSDoc segue o modelo que, conceptualmente, melhor responde ao problema identificado neste trabalho ao permitir a documentação de alto nível, por adopção de um método *multiple-source* baseado na tecnologia wiki, garantindo a consistência semântica entre conteúdos, e possibilitando a produção colaborativa de documentação directamente em suporte web.

Após esta análise do estado da arte das técnicas de documentação de software actualmente documentadas na literatura, é possível concluir que o problema identificado neste trabalho não foi ainda resolvido e que a sua solução seria uma mais valia para a documentação ágil de software, bem como para a documentação de software em geral.

3.3 Resultados esperados

Este trabalho tem como objectivo o desenvolvimento de uma ferramenta que permita colmatar todos os problemas identificados na Secção 3.1. Em concreto a ferramenta deverá:

- auxiliar a escrita de documentação externa de software em suporte wiki garantindo a preservação da consistência semântica entre artefactos utilizados na documentação produzida;
- ser extensível por forma a permitir uma fácil evolução no sentido de se tornar escalável no que diz respeito ao tipo de projectos e linguagens suportados;
- ser totalmente independente do sistema wiki utilizado;
- ser extremamente fácil de adoptar e utilizar.

Esta ferramenta tem como objectivo final o incentivo à documentação de projectos de software em fases ainda prematuras do ciclo de desenvolvimento, apostando, para tal, na maior abrangência possível do tipo de projectos que possam tirar proveito da sua existência e na sua facilidade de adopção e utilização.

A ferramenta será fortemente baseada no conceito da transclusão (Nelson, 1965) de artefactos de software para a preservação da sua consistência semântica na documentação produzida.

O resultado esperado é uma ferramenta funcional que permita validar todos os objectivos aqui delineados.

3.4 Método de investigação

A engenharia de software tem uma história relativamente curta e está ainda em amadurecimento como uma área de investigação. Que métodos de investigação utilizar em pesquisas de engenharia de software é ainda um assunto em debate e é, por si só, uma questão em aberto dentro da comunidade de pesquisa de engenharia de software (Tichy et al., 1993; Zerkowitz and Wallace, 1998).

O desenvolvimento de software tem características específicas que sugerem seu próprio paradigma de investigação, combinando aspectos de outras disciplinas: é um fenómeno criativo humano; projectos de software são caros e geralmente têm ciclos de vida longos; é difícil controlar todos os parâmetros relevantes; a tecnologia muda rapidamente e, como tal, os conhecimentos adquiridos tornam-se rapidamente obsoletos; é difícil de reproduzir estudos; e há poucas teorias de base comum. (Aguiar, 2003)

A categorização proposta no workshop de Dagstuhl (Tichy et al., 1993), os grupos investigaram métodos em quatro categorias genéricas, citando (Zerkowitz and Wallace, 1998):

Método científico “Os cientistas desenvolvem uma teoria para explicar um fenómeno; propõem uma hipótese e, em seguida, testam variações alternativas da hipótese. Durante esse processo, colecionam dados para verificar ou refutar as alegações da hipótese.”

Método de engenharia “Os engenheiros desenvolvem e testam uma solução para uma hipótese. Com base nos resultados do teste, melhoram a solução até que não sejam necessárias mais melhorias.”

Método empírico “Um método estatístico é proposto como uma forma de validar uma dada hipótese. Ao contrário do método científico, pode não haver um modelo formal ou uma teoria que descreva a hipótese. Os dados são colecionados para verificar a hipótese.”

Método analítico “Uma teoria formal é desenvolvida, e os resultados derivados dessa teoria podem ser comparados com observações empíricas.”

Estas categorias aplicam-se a ciências em geral. A experimentação em engenharia de software requer abordagens mais específicas. Como a investigação em engenharia de software engloba questões de ciência da computação, questões humanas e questões organizacionais, muitas vezes, é conveniente usar combinações de abordagens de investigação, tanto de ciências da computação como de ciências sociais.

O método de investigação utilizado nesta dissertação foi uma aproximação ao *método de engenharia*, com base em *casos de estudo* por forma a permitir iterativamente refinar a solução com novas hipóteses extraídas das soluções entretanto desenvolvidas.

Capítulo 4

Projectos exploratórios

Os primeiros projectos exploratórios desenvolvidos para este trabalho foram realizados no contexto do projecto de investigação Doc-It! (FEUP/DEEC, 2006a), em 2006. Os projectos consistiram em adaptar o XSDoc a outros wikis para além do original (VeryQuickWiki (Cronin and Barnett, 2006)), em concreto, pretendeu-se integrar as funcionalidades únicas do XSDoc nos wikis SnipSnap e MoinMoin (Hermann and Waldmann, 2009; Jugel and Schmidt, 2003). Os projectos tiveram os seguintes objectivos principais:

- Desacoplar as funcionalidades do XSDoc do seu wiki base, possibilitando assim reutilizar o melhor do XSDoc – sincronismo de conteúdos, enriquecimento de recursos (como código formatado e navegável) e a sua integração em ambientes de desenvolvimento integrados – noutros wikis que oferecessem funcionalidades igualmente importantes à escrita de documentação de software tais como a gestão de utilizadores, o controlo de acessos a documentos, templates, entre outros;
- Evoluir o sistema de sincronização por forma a permitir a sincronização da documentação produzida com artefactos provenientes de sistemas de controlo de versões (CVS (Free Software Foundation, 1998), Subversion (CollabNet, 2009)), normalmente utilizados nos projectos de desenvolvimento;
- Evoluir funcionalidades de processamento de código-fonte para uma representação mais eficaz na documentação;
- Suportar outras linguagens de programação.

Os projectos desenvolvidos foram denominados de *XSDoc for SnipSnap* e *XSDoc for MoinMoin* (FEUP/DEEC, 2006c; FEUP/DEEC, 2006b).

4.1 XSDoc for SnipSnap

As principais funcionalidades do XSDoc foram inicialmente integradas no wiki SnipSnap uma vez tratar-se de um wiki com funcionalidades interessantes para a documentação de software e partilhar a base tecnológica Java com o XSDoc original.

Dada a importância do controlo de acessos a áreas específicas do wiki, foi desenvolvido no âmbito deste projecto exploratório, um sistema, por extensão ao SnipSnap original, para controlo preciso de acesso a utilizadores. A Figura 4.1 representa a utilização da extensão desenvolvida para a configuração de um utilizador no SnipSnap. O sistema de controlo de acesso desenvolvido é baseado em papéis (*roles*) e grupos.

The screenshot shows a web application window titled "SnipSnap :: Setup :: User Management". The interface is divided into two main sections. On the left, a sidebar titled "Select a Configuration Area" lists various settings: Basic Settings, Theme Selection, Localization, Permissions, Email Settings, Mobile Blogging, Proxy Server, Expert Settings, **User Management** (highlighted), Group Management, Import Database, Export Database, Search Engine, and Maintenance. Below this list is a dashed box containing the text: "The user management allows to add, modify and delete users from your system." The main content area is titled "User Management" and contains several form fields and sections. It includes labels for "The login name of the user.", "The password used to login.", "The users Email address. This address is used to contact the user in case of notifications and password changes.", "The roles of this user.", "The groups of this user.", and "The status is a simple text that can be used to mark the user. It is displayed in the user list." The form fields include "User Name" (with the value "Nuno Baldaia"), "Password", "Password again", "Email", "Roles" (with checkboxes for Admin, Editor, NoComment, and Upload, where Admin and Editor are checked), "Groups" (with checkboxes for config, MAD5-2007-08, e-ASLA, Fenix, and Private), and "Status" (with the value "not set"). At the bottom right of the form are "Save User" and "Cancel" buttons. The footer of the window displays "SnipSnap 1.0b2-feup-v8".

Figura 4.1: Sistema de gestão de utilizadores para o SnipSnap (FEUP/DEEC, 2006c)

Para a configuração de projectos de documentação, nomeadamente a configuração do acesso ao sistema de controlo de versões e as configurações dos projectos de software Java a documentar, são utilizadas páginas (*snips*) convencionais que podem ser devidamente protegidas pelo sistema de controlo de acessos desenvolvido. A Figura 4.2 representa um *snip* de configuração de um projecto Java disponível a partir de um sistema de controlo de versões Subversion.

A inclusão de recursos é feita através de três macros específicas: *source*, *source-link* e *javaml-uml*.

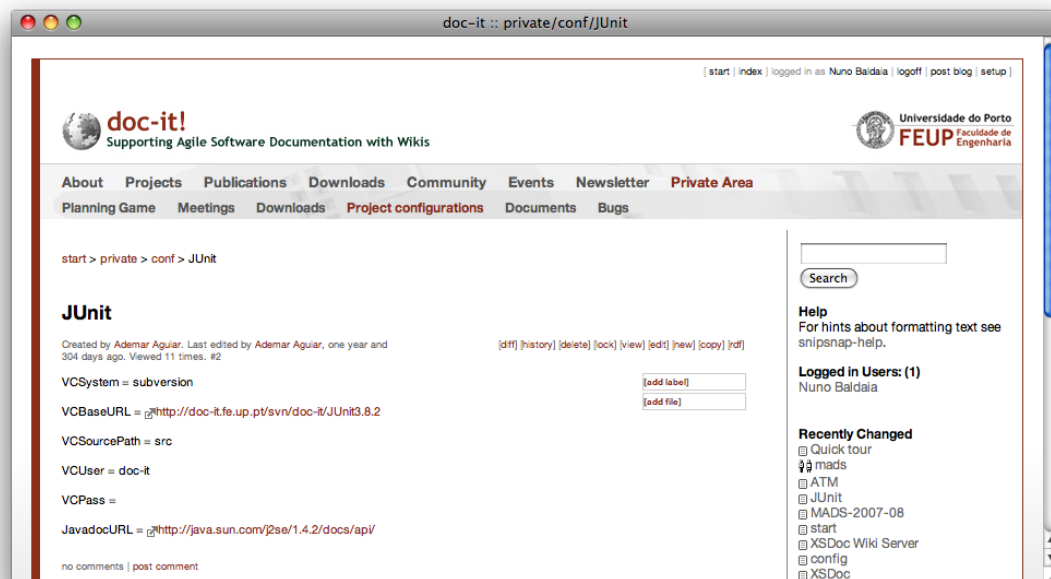


Figura 4.2: Configuração de um projecto de documentação no SnipSnap (FEUP/DEEC, 2006c)

A macro *source* possibilita a inclusão de fragmentos de código-fonte formatado. Particularmente para a linguagem Java, esta macro possibilita a inclusão de fragmentos de código-fonte navegável, utilizando para tal o sistema de processamento de código-fonte do XSDoc original baseado na representação JavaML de código Java. A Figura 4.3 representa um exemplo da utilização da macro *source* no SnipSnap.

A macro *source-link* tem as mesmas funcionalidades da macro *source* mas os recursos representados são acessíveis por hiperligação em vez de serem incluídos na página.

A macro *javaml-uml* possibilita a inclusão de diagramas UML automaticamente gerados a partir de código-fonte Java. A geração dos diagramas é feita com recurso ao sistema UMLGraph (Spinellis, 2006) que possibilita a especificação declarativa e desenho de diagramas de classes e de sequência UML. Os diagramas são gerados na especificação de diagramas Graphviz (Graphviz, 2009) que faz a geração de diagramas a partir de descrições de gráficos numa linguagem simples de texto, em vários formatos úteis como imagens e SVG (Ferraiolo et al., 2003) para páginas web, Postscript para inclusão em PDF ou outros documentos, ou exibição em browsers gráficos interactivos. A Figura 4.4 representa um exemplo da utilização da macro *javaml-uml* no SnipSnap.

O sistema de processamento de código-fonte baseado na representação JavaML do XSDoc original utiliza uma versão estendida do Jikes (IBM, 2003), um compilador de código Java escrito em C++, para a transformação de código Java na representação JavaML (Aguiar et al., 2004). Este compilador de JavaML foi a primeira dificuldade encontrada na evolução do XSDoc para o SnipSnap por dois motivos:

1. O Jikes é um compilador obsoleto, o que compromete seriamente a evolução do compilador JavaML face à evolução da linguagem Java;

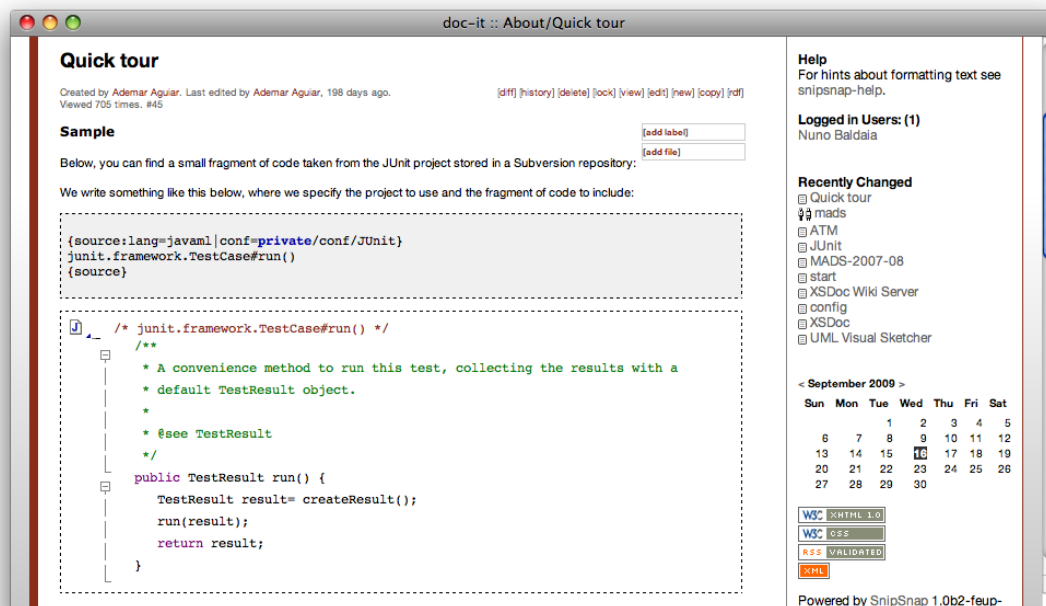


Figura 4.3: Macro *source* no wiki SnipSnap (FEUP/DEEC, 2006c)

2. A compilação de código Java para JavaML requer a presença do projecto e de todas as suas configurações (classpath, sourcepath, etc.), pelo que a integração do XSDoc com sistemas de controlo de versões para acesso ao código constitui uma dificuldade acrescida.

Dadas estas dificuldades de conversão de código Java para JavaML, considerou-se que a representação do código Java em JavaML estaria já disponível no sistema de controlo de versões remoto e que seria utilizada pelo XSDoc para inclusão de fragmentos de código Java na documentação.

Como resultado final, foi obtida uma nova versão do XSDoc com o wiki base substituído pelo SnipSnap, capaz de usar código proveniente de sistemas de controlo de versões Subversion, o que constitui uma evolução no aspecto da preservação da consistência semântica entre recursos utilizados para a documentação. No entanto, as funcionalidades de processamento de código Java navegável dependem da presença da representação do código-fonte em JavaML no sistema de controlo de versões, o que dificulta bastante a facilidade de adopção desta ferramenta, que é um dos principais objectivos delineados para este projecto.

Não houve, no entanto, grandes avanços relativamente ao XSDoc no que diz respeito à flexibilidade para suporte de diferentes tipos de recursos, assim como na independência do suporte de autoria de documentação, uma vez que obriga à utilização do SnipSnap. A facilidade de adopção foi, ainda assim, melhorada uma vez que depende apenas da instalação de plugins no SnipSnap e da presença do cliente Subversion no servidor do wiki.

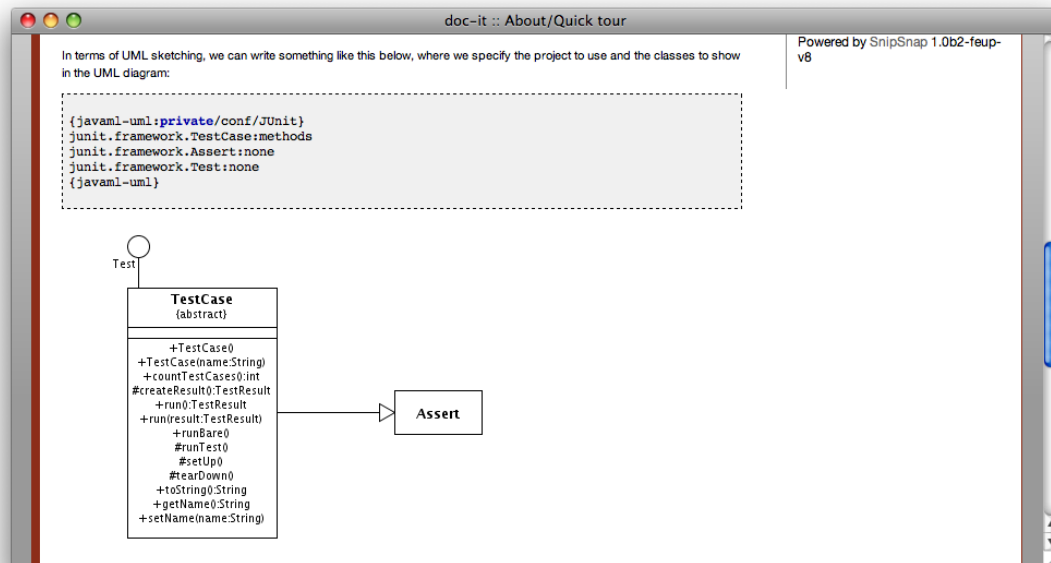


Figura 4.4: Macro *javaml-uml* no wiki SnipSnap (FEUP/DEEC, 2006c)

4.2 XSDoc for MoinMoin

No projecto exploratório seguinte, desenvolveu-se uma adaptação do XSDoc para o wiki MoinMoin. Nesta adaptação optou-se por substituir o complexo e poderoso mecanismo de processamento para a geração da representação navegável em HTML de código-fonte Java a partir da sua representação em JavaML, proveniente do XSDoc original, por um simples interpretador de código Java baseado no ANTLR (Parr, 2009). Este novo mecanismo permite simplesmente obter fragmentos de código para inclusão na documentação ficando a formatação sintáctica de código a cargo do próprio MoinMoin. Com esta simplificação, abdicou-se da capacidade de navegação no código através de hiperligações automaticamente inseridas pelo JavaML com base em aspectos sintácticos e semânticos dos programas compilados.

Esta versão permite também o acesso a recursos provenientes de sistemas de controlo de versões Subversion. No entanto, tentou-se evoluir o sistema experimentado no SnipSnap por forma a ser possível compilar código no próprio servidor do wiki. Para tal, optou-se por descarregar revisões integrais dos projectos para o servidor do wiki.

O sistema é todo encapsulado na forma de uma macro – *Resource Macro* – com diversas vistas para diferentes fins, nomeadamente as vistas *config*, *sourcecode*, *text*, *image*, *uml* e *dml*.

A configuração dos projectos de documentação é feita com recurso à macro *Resource Macro* na vista *config* e é persistida em páginas wiki. Esta vista disponibiliza um formulário web para configuração do acesso a sistemas de controlo de versões Subversion, configuração de projectos Java para possibilitar a compilação de código-fonte (*source path*, *class path*); e gestão de revisões do projecto disponíveis no sistema. A Figura 4.5 representa a vista de configuração de um projecto de documentação no MoinMoin.

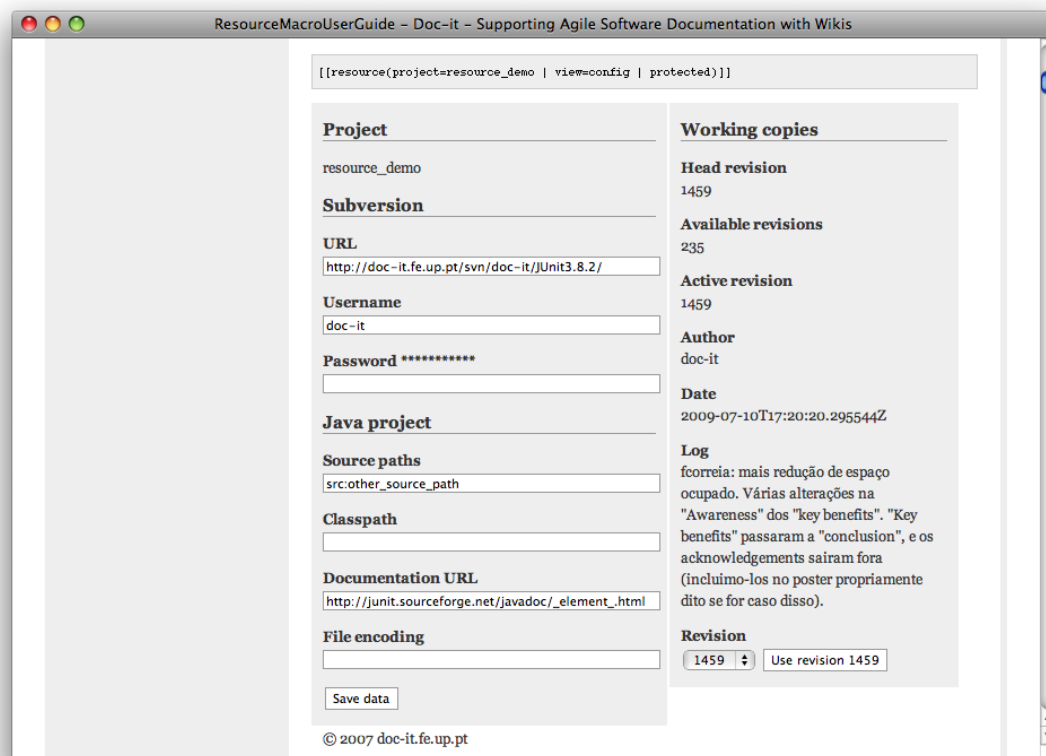


Figura 4.5: Configuração de um projecto de documentação no MoinMoin (FEUP/DEEC, 2006b)

A vista *sourcecode*, vista activa por omissão, permite a inclusão de fragmentos de código-fonte Java em páginas wiki. Esta vista tem um comportamento semelhante à macro *source* do SnipSnap para a linguagem *javaml*, à excepção das funcionalidades de navegação no código que não estão presentes nesta macro, uma vez que a compilação da representação do código-fonte em JavaML foi abandonada neste projecto, tal como foi já referido. A formatação sintáctica do código apresentado é feita pelo próprio mecanismo de formatação de código do MoinMoin. A Figura 4.6 representa alguns exemplos da utilização da vista *sourcecode* para inclusão de fragmentos de código-fonte Java numa página wiki.

A vista *text* permite incluir qualquer tipo de recurso de texto em páginas wiki. A formatação sintáctica dos recursos é feita pelo próprio mecanismo de formatação sintáctica de código do MoinMoin, sendo, para tal, necessário fornecer à macro a informação relativa à linguagem do recurso por meio de um parâmetro.

A vista *image* permite incluir imagens disponíveis no Subversion nos documentos produzidos.

A vista *uml* tem um comportamento similar ao da macro *javaml-uml macro* do projecto *XSDoc for SnipSnap*. Concretamente, permite a inclusão de diagramas UML automaticamente gerados a partir do código fonte Java, disponível localmente no servidor do wiki, utilizando também para tal a biblioteca UMLGraph. A Figura 4.7 representa um exemplo da utilização da vista *uml* para geração e inclusão de um diagrama de classes para três classes da framework JUnit. A listagem

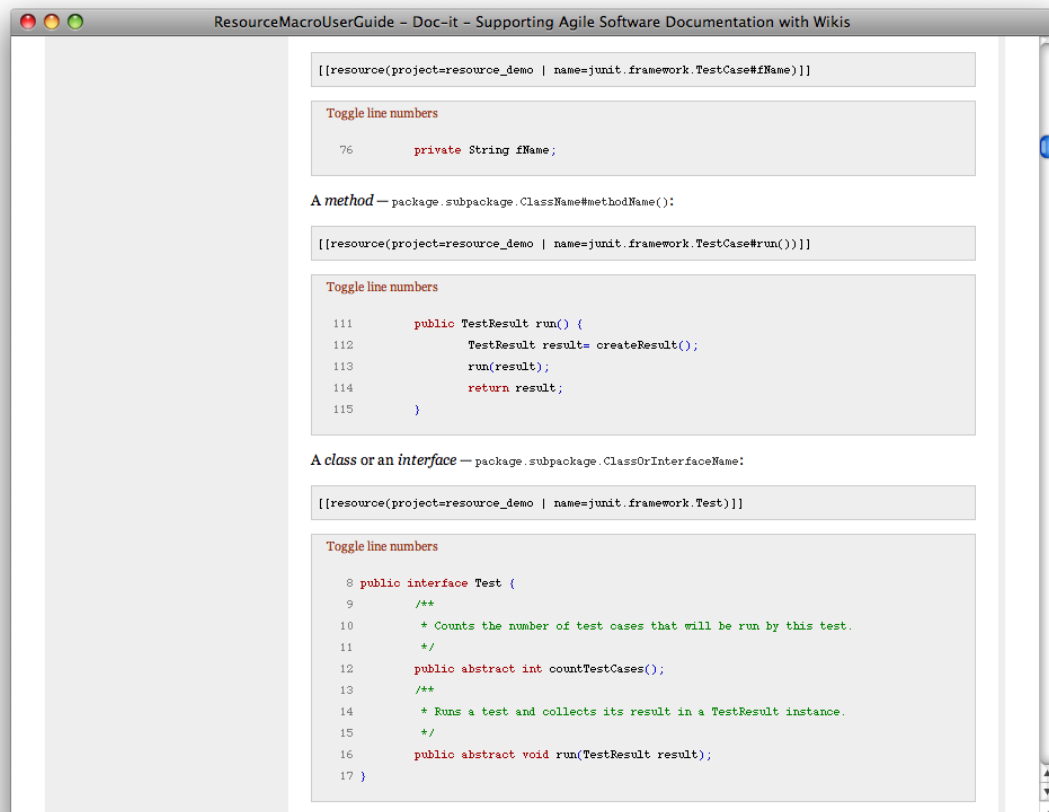


Figura 4.6: Vista *sourcecode* da *Resource Macro* do MoinMoin (FEUP/DEEC, 2006b)

abaixo contém a utilização da macro que deu origem ao diagrama gerado, representado na figura.

```
[[resource(project=resource_demo | view=uml |
name=junit/samples/money/MoneyTest.java
junit/framework/TestCase.java junit/framework/Test.java)]]
```

Relativamente ao SnipSnap, perdeu-se neste projecto exploratório a capacidade de inclusão de código navegável em consequência do abandono do compilador de Java para JavaML do XSDoc original. No entanto, conseguiu-se um ambiente potencialmente favorável à compilação de código localmente no servidor do wiki, o que permitiria transferir para o wiki a responsabilidade de processamento de recursos e, consequentemente uma maior facilidade de utilização.

Ambos os projectos exploratórios acabaram por ser abandonados uma vez que as soluções experimentadas não resolviam alguns dos principais objectivos delineados para este trabalho, nomeadamente a facilidade de adopção, facilidade de utilização, independência relativamente aos recursos utilizados e independência relativamente ao sistema de documentação wiki utilizado.

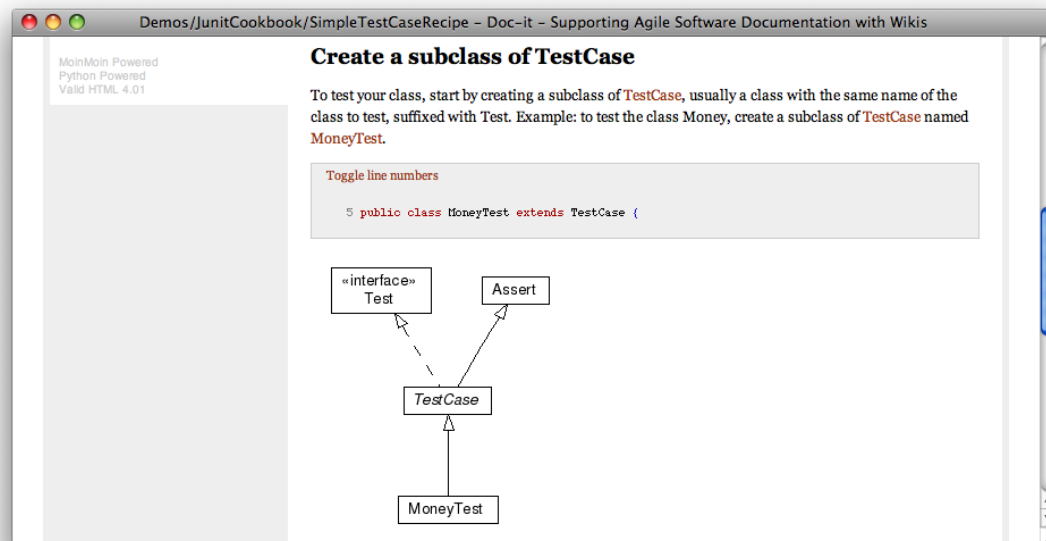


Figura 4.7: Exemplo da utilização da vista *uml* da macro Resource Macro do MoinMoin (FEUP/DEEC, 2006b)

4.3 Adaptação do XSDoc a outros wikis

A arquitectura base utilizada em ambos os projectos exploratórios de adaptação do XSDoc a outros wikis foi a mesma. A lógica do sistema é colocada sob a forma de macros (*XSDoc macro*) que estendem cada um dos wikis. Estas macros responsabilizam-se por aceder ao sistema de controlo de versões para extracção dos recursos necessários, processá-los utilizando o módulo de processamento de recursos (*XSDoc tools*), e seguidamente disponibilizam os conteúdos ao wiki que serve como ferramenta de visualização e autoria de documentação.

A Figura 4.8 representa a arquitectura utilizada nos projectos exploratórios *XSDoc for SnipSnap* e *XSDoc for MoinMoin*.

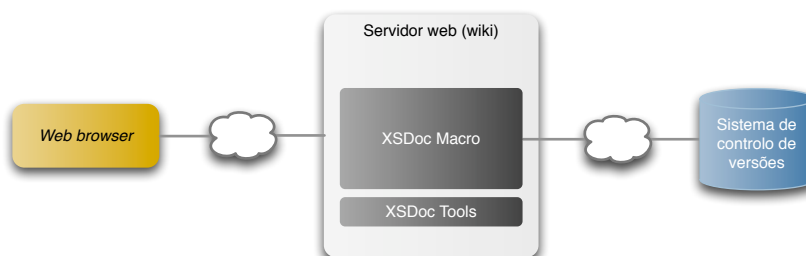


Figura 4.8: Arquitectura utilizada nos projectos exploratórios *XSDoc for SnipSnap* e *XSDoc for MoinMoin*

A adaptação do XSDoc a estes dois wikis foi suficiente para se perceber que a adaptação a outros wikis segundo esta arquitectura seria difícil de escalar por obrigar a reescrever grande parte

do código do sistema (*macros*) para cada wiki em concreto, o que seria extremamente trabalhoso e limitaria a sua evolução.

Este problema seria evitável alterando ligeiramente a arquitectura do sistema por forma a que todo o processamento independente do wiki fosse transferido para módulos reutilizáveis nos diversos wikis – *XSDoc Processing Module* e *XSDoc Tools* – que seriam responsáveis pelo acesso ao sistema de controlo de versões e processamento de recursos, o que tornaria a macro específica de cada wiki – *XSDoc Macro* – mais leve e fácil de implementar. A Figura 4.9 representa uma possível arquitectura.

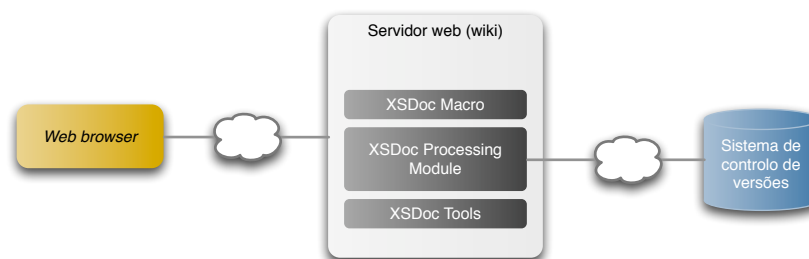


Figura 4.9: Arquitectura para adaptação do XSDoc aos vários wikis

Muito embora uma arquitectura deste género permitisse facilitar a adaptação do sistema a diferentes wikis, a sua instalação seria sempre uma tarefa não trivial, uma vez que são utilizadas diversas tecnologias, o que constitui, ainda assim, um substancial entrave à sua fácil adopção em grande escala.

Estas experiências foram de extrema importância para se descobrirem os potenciais problemas de um sistema de documentação que pretenda suportar documentação externa, segundo os métodos *multiple-source*, mantendo a consistência semântica entre recursos utilizados, o que motivou o abandono das abordagens aqui experimentadas a favor da implementação de um sistema totalmente novo, desenvolvido de raiz.

Capítulo 5

CrystalBox

Com base na experiência acumulada na concepção e desenvolvimento dos projectos exploratórios anteriormente referidos, para prossecução dos objectivos deste trabalho, evoluiu-se para o desenvolvimento de uma nova solução para apoiar a documentação de software – a ferramenta CrystalBox.

A ferramenta CrystalBox foi desenvolvida de raiz seguindo o conceito base da infra-estrutura XSDoc. Totalmente baseada na web, a ferramenta tenta resolver os problemas de falta de flexibilidade e dificuldade de adopção encontrados no XSDoc.

5.1 Objectivos

Os principais objectivos delineados para a ferramenta CrystalBox foram os seguintes:

- Disponibilizar uma ferramenta de documentação facilmente integrável em qualquer wiki;
- Permitir acesso fácil, síncrono e transparente aos artefactos de projectos de software para inclusão na documentação produzida, seguindo um método *multiple-source*;
- Permitir o enriquecimento de artefactos de software, como por exemplo, formatação sintáctica de código-fonte;
- Ser facilmente extensível por forma a suportar diferentes realidades de desenvolvimento de software, tais como, diferentes linguagens de programação ou diferentes ambientes de desenvolvimento;
- Ser extremamente fácil de adoptar e utilizar.

A ferramenta de documentação CrystalBox pretende incentivar a escrita de documentação ainda em fases prematuras do desenvolvimento dos projectos de software. Para tal, a ferramenta pretende oferecer um sistema extremamente fácil de adoptar e utilizar, que viabilize a criação expedita de documentos com referências para artefactos externos dos projectos de desenvolvimento, e garanta a sua constante actualização ao longo de todo o ciclo de desenvolvimento. A ferramenta pretende

ainda ser suficientemente abrangente no que diz respeito às diferentes realidades dos projectos de desenvolvimento, devendo disponibilizar, para tal, mecanismos de extensibilidade.

5.2 Requisitos

A ferramenta CrystalBox pressupõe uma realidade característica do desenvolvimento colaborativo de software, típico das metodologias ágeis, ou do desenvolvimento *open-source* no qual a equipa, ou comunidade, desenvolve e publica recursos para um sistema de controlo de versões. A Figura 5.1 representa os principais casos de utilização da ferramenta para esta realidade.

No caso típico de utilização, o autor da documentação, também ele programador, dever-se-á registar na aplicação web CrystalBox BaseStation para criar e configurar um novo projecto de documentação, relativo ao projecto de software que pretende documentar.

O projecto de documentação, uma vez devidamente configurado, oferece à aplicação CrystalBox BaseStation a autonomia suficiente e necessária para aceder aos recursos disponíveis no sistema de controlo de versões.

O autor da documentação escreve documentos com simples referências para recursos do projecto, podendo, para tal, contar com a ajuda de wikis com macros dedicadas. No cliente, *web browser*, os recursos referenciados são automaticamente inseridos no documento pela camada de cliente da ferramenta – CrystalBox Client, de forma síncrona e transparente.

O programador acede à documentação do sistema que inclui recursos do projecto de software (artefactos) que se mantêm actualizados ao longo de todo o processo de desenvolvimento, uma vez que o projecto de desenvolvimento e a documentação utilizam o mesmo sistema de controlo de versões. Assim, as alterações feitas aos recursos são automaticamente repercutidas na documentação produzida através das referências utilizadas, sem qualquer necessidade de actualização.

5.3 Arquitectura lógica e física

A ferramenta CrystalBox segue a convencional arquitectura de três camadas para satisfazer os objectivos delineados:

1. Camada de dados – sistema de controlo de versões;
2. Camada applicacional – aplicação web CrystalBox BaseStation;
3. Cliente – CrystalBox Client.

A Figura 5.2 representa a arquitectura da ferramenta CrystalBox. Embora não faça parte da arquitectura da CrystalBox, o servidor web, que tipicamente serve as páginas de documentação (wiki), foi representado na figura para uma melhor compreensão do funcionamento da ferramenta.

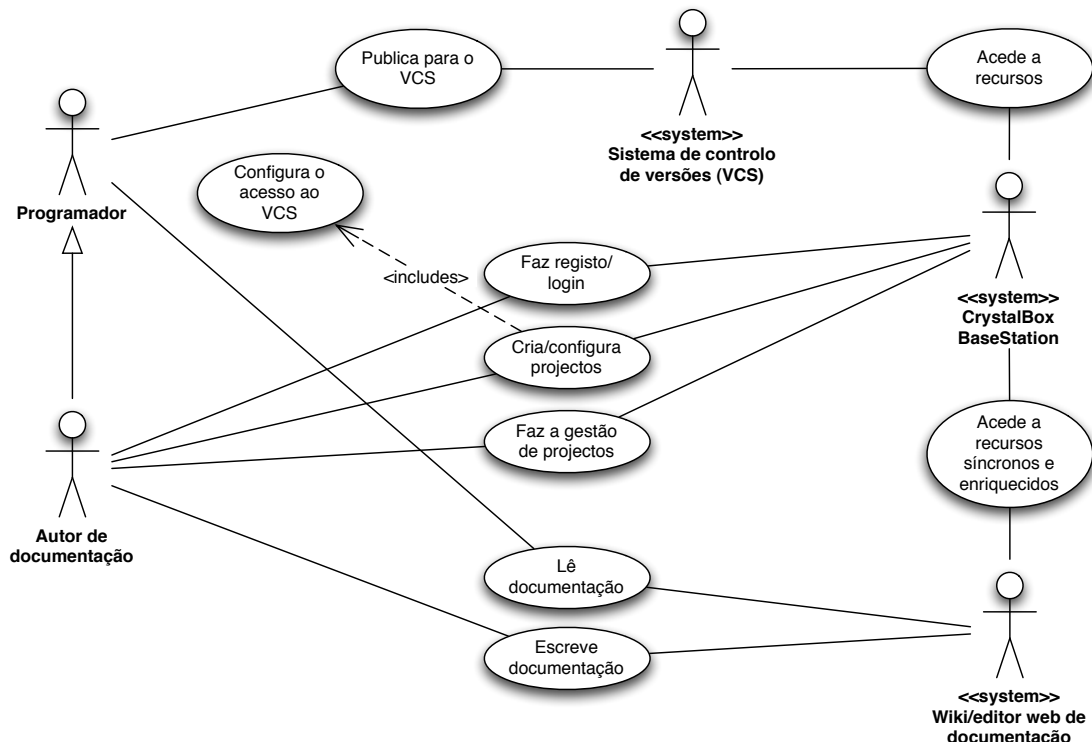


Figura 5.1: Principais casos de utilização da ferramenta CrystalBox

5.3.1 Camada de dados

A grande maioria de projectos de software recorrem a sistemas de controlo de versões, como o CVS (Free Software Foundation, 1998), o Subversion (CollabNet, 2009) ou, mais recentemente, o Git (Git, 2009), para armazenamento e controlo de versões dos artefactos constituintes dos projectos de desenvolvimento de software. Os sistemas de controlo de versões são representações perfeitas dos recursos dos projectos uma vez que permitem não só o acesso a todos os seus recursos, mas também a todas as suas versões ao longo do tempo, o que pode ser bastante interessante para a documentação da evolução de um projecto. Uma vez que estes sistemas são normalmente acessíveis por web, constituem uma solução para o acesso ubíquo aos recursos (artefactos) dos projectos, solução ideal para a sua utilização na produção de documentação.

5.3.2 Camada aplicacional

A aplicação web CrystalBox BaseStation foi desenhada por forma a centralizar na camada aplicacional grande parte do trabalho da ferramenta, retirando, desta forma, praticamente todo o peso da camada do cliente.

A aplicação tem como principais funções:

- permitir o registo de utilizadores para acesso a uma área privada de gestão de projectos onde pode ser configurado, para cada projecto, o acesso ao respectivo sistema de controlo

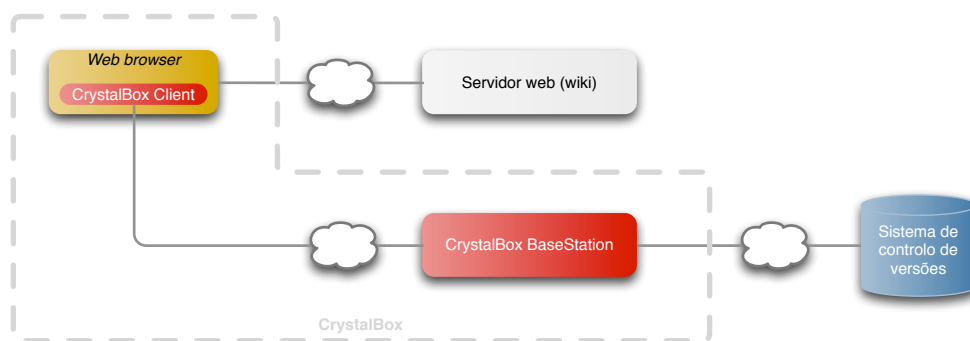


Figura 5.2: Arquitectura física da ferramenta CrystalBox

de versões;

- fornecer de forma transparente acesso aos recursos existentes na camada de dados;
- processar recursos provenientes da camada de dados (ex.: processamento sintáctico de código para visualização com cores; processamento semântico de código para identificação e extracção de fragmentos; geração de diagramas UML; etc.).

Resumidamente, esta camada pretende fornecer à camada de cliente, de forma transparente, recursos disponíveis na camada de dados devidamente processados e enriquecidos.

5.3.3 Cliente

No cliente – *web browser* – funciona a última camada da ferramenta – CrystalBox Client, responsável por apresentar na página de documentação os recursos disponibilizados pela camada aplicacional.

Desta forma, é eliminada a necessidade de qualquer intervenção da arquitectura da CrystalBox no wiki (ou qualquer outra aplicação web) de suporte à produção da documentação.

Esta arquitectura permite, desta forma, cumprir o objectivo da independência da ferramenta relativamente ao wiki utilizado.

5.4 Produto

Do ponto de vista do utilizador, a ferramenta CrystalBox divide-se em dois componentes: a aplicação web CrystalBox BaseStation e a camada de cliente CrystalBox Client.

5.4.1 CrystalBox BaseStation

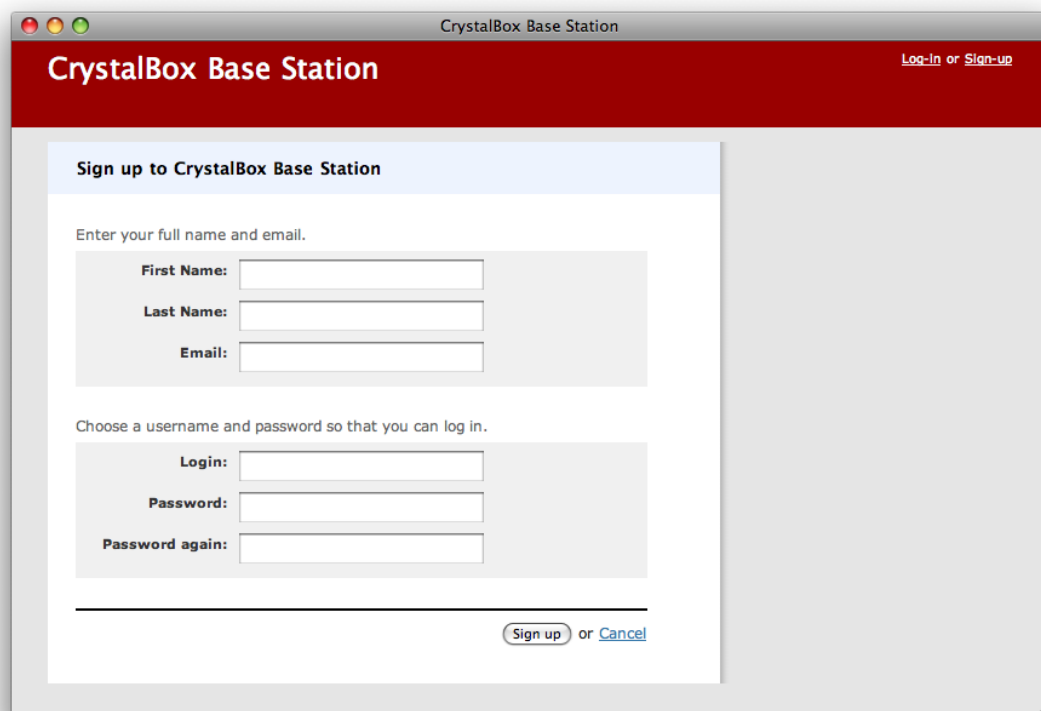
A CrystalBox BaseStation é uma aplicação web que centraliza todo o processamento da ferramenta para o fornecimento de recursos síncronos e enriquecidos para utilização na documentação de software.

A aplicação tem como principais funcionalidades:

- Registo de utilizadores;
- Gestão e configuração de projectos de documentação;
- Fornecimento de recursos existentes em sistemas de controlo de versões pré-processados e sintacticamente formatados à camada de cliente – CrystalBox Client.

Registo de utilizadores

A CrystalBox BaseStation permite o registo de novos utilizadores para que possam criar e configurar projectos de documentação e assim utilizar a ferramenta CrystalBox para a produção de documentação de software. A Figura 5.3 representa o painel de registo de novos utilizadores da CrystalBox BaseStation.



The screenshot shows a web browser window titled "CrystalBox Base Station". The page has a red header bar with the text "CrystalBox Base Station" on the left and "Log-In or Sign-up" on the right. Below the header, there is a light blue box with the title "Sign up to CrystalBox Base Station". Inside this box, the text "Enter your full name and email." is followed by three input fields: "First Name:", "Last Name:", and "Email:". Below these, the text "Choose a username and password so that you can log in." is followed by three input fields: "Login:", "Password:", and "Password again:". At the bottom of the form, there is a "Sign up" button and a "Cancel" link.

Figura 5.3: CrystalBox BaseStation – Registo de utilizadores

Gestão e configuração de projectos de documentação

Cada utilizador registado pode criar vários projectos de documentação associados à sua conta, tendo cada projecto um identificador único em todo o sistema, escolhida pelo próprio utilizador na altura da sua criação. A figura 5.4 representa a lista de projectos de um utilizador.

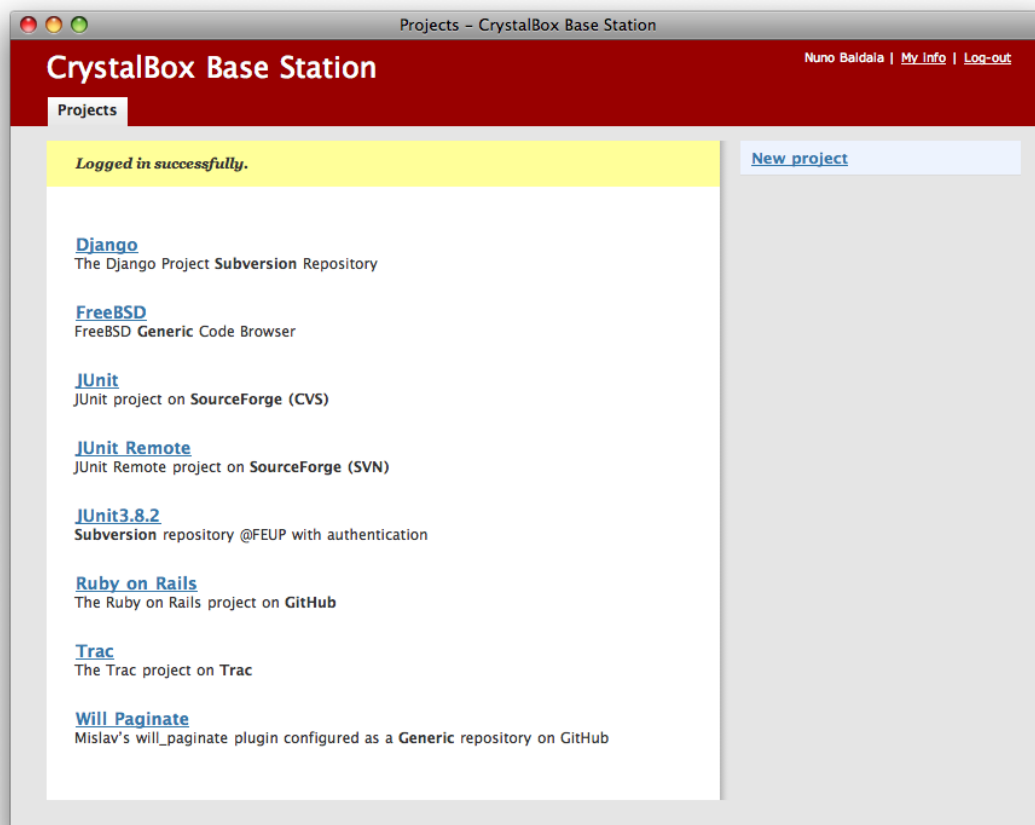


Figura 5.4: CrystalBox BaseStation – Lista de projectos

Um projecto de documentação na aplicação CrystalBox BaseStation consiste simplesmente num conjunto de configurações necessárias ao sistema para o fornecimento de recursos disponíveis em sistemas de controlo de versões acessíveis pela web, e seu processamento.

Acesso a sistemas de controlo de versões

O processo de configuração de projectos envolve a configuração do acesso a um sistema de controlo de versões acessível por web. Actualmente a aplicação suporta o acesso directo a sistemas de controlo de versões Subversion (SVN) (CollabNet, 2009) com possibilidade de autenticação e, potencialmente, a qualquer sistema de controlo de versões aberto com interface web, como por exemplo, sistemas com interface ViewVC (ViewVC, 2009) para exploração de código. A aplicação disponibiliza pré-configurações para acesso aos serviços GitHub (Awesome, 2009) e SourceForge (SourceForge, 2009), bem como para as aplicações *Trac* (Software, 2009). A Figura 5.5 representa o painel de configuração do sistema de controlo de versões de um projecto.

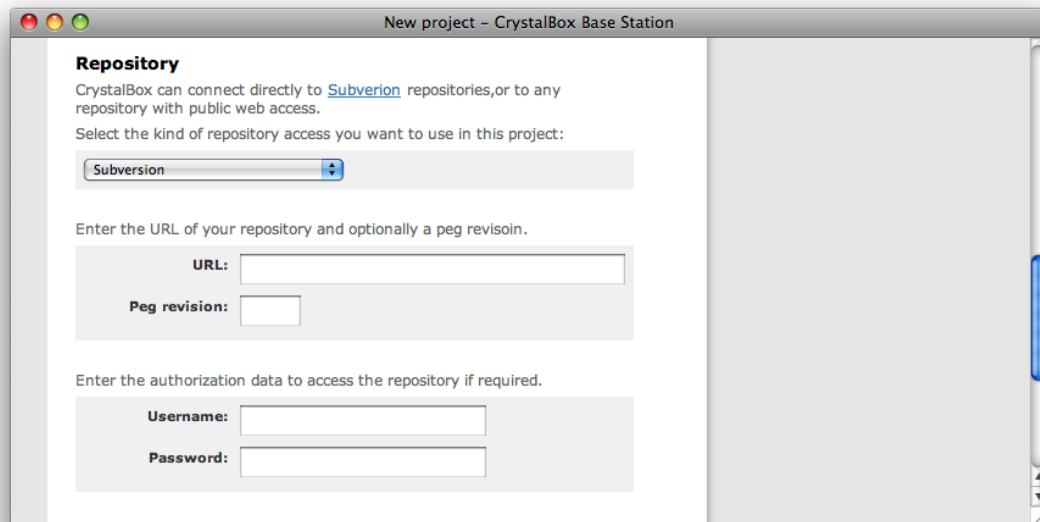


Figura 5.5: CrystalBox BaseStation – Configuração do sistema de controlo de versões

Fornecimento de recursos à camada de cliente

A aplicação CrystalBox BaseStation é também responsável pelo processamento de recursos para, por exemplo, formatação sintáctica e processamento ao nível semântico de código, para fornecimento à camada de cliente.

O sistema está preparado para receber módulos de processamento específicos para diferentes tipos de recursos, suportando actualmente, processamento de código-fonte nas linguagens Java e Ruby através de interpretadores semânticos de código.

Todos os recursos de código-fonte fornecidos pelo sistema são processados sintacticamente para visualização com cores, independentemente do pré-processamento de que possam ser alvo ao nível semântico.

Os recursos binários do tipo imagem são directamente fornecidos pela aplicação para visualização na documentação produzida. Para outros recursos binários (que não imagens), são disponibilizados na documentação em forma de links para que possam ser descarregados.

5.4.2 CrystalBox Client

A ferramenta disponibiliza um ficheiro JavaScript¹ que deverá ser incluído na página de documentação. Este ficheiro contém toda a lógica da camada do cliente, responsável pelo processamento das referências aos recursos provenientes da camada aplicacional para inclusão na documentação.

¹Disponível em <http://crystalbox.org/crystalbox.js>

Desta forma, a plataforma de documentação poderá ser *qualquer sistema que permita a edição de páginas HTML* no qual seja possível incluir um *script* externo, o que viabiliza a utilização da CrystalBox noutros suportes para além dos wikis.

Os wikis poderão utilizar a CrystalBox simplesmente através da escrita directa de código HTML, caso esta seja permitida, ou poderão, preferencialmente, facilitar ainda mais a sua utilização através da disponibilização de pequenas extensões (*macros*) que permitam simplificar a referência dos recursos. Uma vez que tais extensões são, de forma geral, extremamente simples de implementar, torna-se viável a extensão de qualquer wiki por forma a permitir e facilitar a escrita de documentação usando a CrystalBox. A utilização da ferramenta não condiciona, portanto, a livre escolha da plataforma de suporte à escrita de documentação mais conveniente para cada caso particular.

O processo de adopção da ferramenta fica assim bastante facilitado, o que constitui um forte incentivo à sua utilização.

Referência a recursos

A referência aos recursos é feita por âncoras HTML (*links*) devidamente identificadas com o atributo `rel="crystalbox"`. O formato do *link* é o seguinte:

```
<a rel="crystalbox"
  href="http://crystalbox.org/project_id/resource_path?arguments">
  texto do link</a>
```

Os *links* cujo atributo `rel` tenha o valor `crystalbox` serão processados como recursos da CrystalBox pelo *script* `crystalbox.js` incluído na página HTML. Este *script* é responsável pelo processamento desses *links* por forma a permitir a incorporação dos recursos referenciados na própria página de documentação sem a necessidade de abrir uma nova página. Os recursos referenciados são visualizados, por omissão, numa caixa – caixa CrystalBox – que abre quando o *link* é visitado (Figura 5.6), podendo, opcionalmente, ser inseridos no corpo do documento, bastando para tal utilizar o atributo `class="inline"` nos *links* de referência para os recursos (Figura 5.7).

Navegação nos recursos

Os recursos podem ser navegáveis, ou seja, é possível que recursos como código-fonte ou diagramas UML referenciem outros recursos. Se tal acontecer, essa navegação será sempre feita na caixa CrystalBox, o que lhe dá um carácter de *caixa mágica* de visualização de recursos remotos. Daí o nome CrystalBox.

As referências para os recursos CrystalBox são *degradable*, ou seja, caso o *web browser* não possa, por qualquer motivo, processar JavaScript, o link funcionará normalmente para o recurso referenciado numa nova página sem qualquer problema. Desta forma, a ferramenta pode, no limite, ser utilizada mesmo em sistemas que não permitam a inclusão do *script* `crystalbox.js`,



Figura 5.6: CrystalBox Client – visualização de um recurso na caixa CrystalBox

para ligação normal a recursos síncronos e enriquecidos fornecidos pela aplicação CrystalBox BaseStation.

Macros para wikis

Os wikis são excelentes ferramentas para a escrita colaborativa de documentação, tal como já foi mencionado anteriormente. Uma vez integrados com a CrystalBox os wikis constituem uma excelente ferramenta de documentação ágil de software.

Os wikis podem ser facilmente estendidos para facilitar a referência de recursos disponibilizados pela ferramenta CrystalBox através de pequenas extensões (*macros* ou *plugins*) que facilitem a escrita de *links* para os recursos e incluam automaticamente o *script* `crystalbox.js` na página HTML de documentação, permitindo a utilização da ferramenta de forma simplificada e transparente por parte do utilizador.

A ferramenta CrystalBox disponibiliza já macros² para os wikis DokuWiki, MediaWiki, Moin-Moin e Trac (DokuWiki, 2009; Foundation, 2009c; Hermann and Waldmann, 2009; Software, 2009), bem como toda a documentação³ necessária para a construção de extensões para novos wikis.

A listagem seguinte contém um exemplo da utilização da macro CrystalBox para o wiki do sistema de gestão de projectos Trac (Software, 2009).

```
[[CrystalBox(
```

²Disponíveis em http://wiki.crystalbox.org/download_wiki_macros

³Disponível em http://wiki.crystalbox.org/how_to_write_a_wiki_macro



Figura 5.7: CrystalBox Client – visualização de um recurso incluído no corpo documento

path=junit/junit/framework/TestCase.java?member=#runBare()]]

O resultado é um *link* para o código do método `runBare()` da classe `TestCase` da framework `JUnit`. Quando acedido, este link abre uma caixa `CrystalBox` com o recurso pretendido. A Figura 5.8 representa o resultado da utilização da macro representada na listagem acima.

5.5 Arquitetura tecnológica

Ao nível da implementação, a ferramenta `CrystalBox` pode também ser dividida em dois sistemas distintos que comunicam entre si:

1. Aplicação web `CrystalBox BaseStation`;
2. A camada de cliente `CrystalBox Client` responsável pelo processamento dos recursos no cliente.

A aplicação `CrystalBox BaseStation` foi desenvolvida sobre a framework de desenvolvimento de aplicações web `Ruby on Rails` (Hansson, 2009).

5.5.1 Ruby on Rails

`Ruby on Rails` é uma framework *open-source* para a linguagem de programação `Ruby` que segue o padrão de arquitetura *Model View Controller* (MVC) (Reenskaug, 1979) e disponibiliza uma série de ferramentas e mecanismos que permitem a criação de aplicações com base em estruturas pré-definidas por forma a aumentar velocidade e facilidade de desenvolvimento.

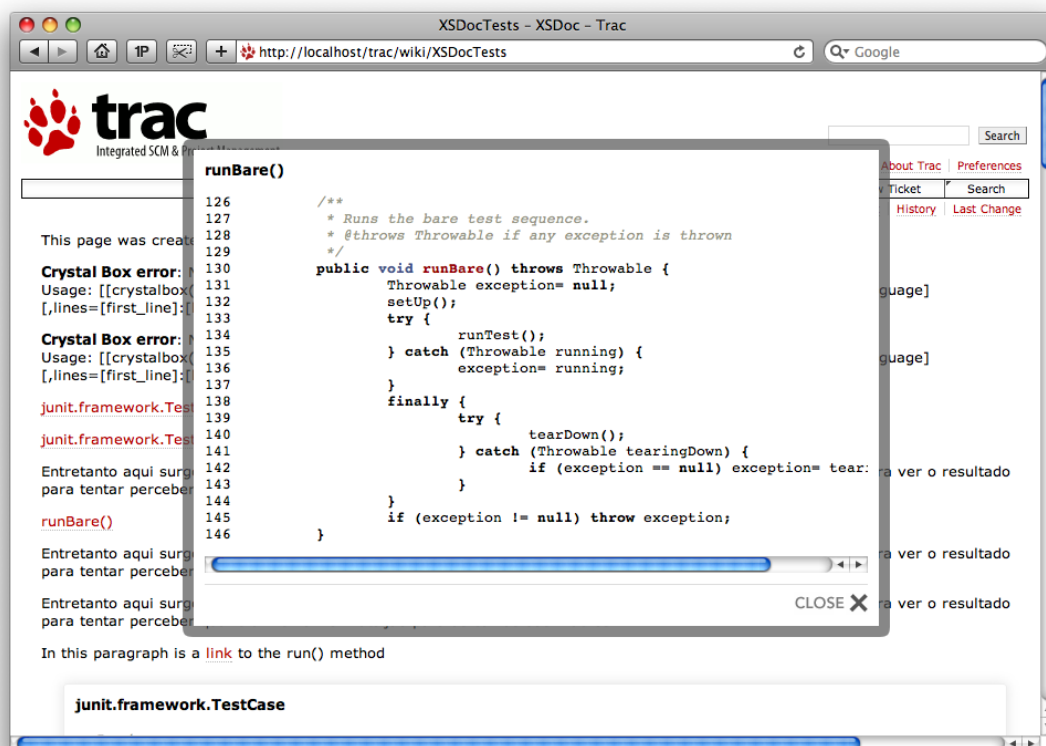


Figura 5.8: Exemplo de utilização da macro CrystalBox no wiki do Trac

O desenho do Rails tem como filosofia base os conceitos *don't repeat yourself* (DRY) e *convention over configuration*. DRY indica que cada parte do conhecimento num sistema deve ser expresso apenas num local. O princípio da convenção sobre configuração significa que o Rails tem predefinições para quase todos os aspectos de uma aplicação. Seguindo as convenções, é possível escrever aplicações Rails usando menos código que o habitual uma vez que são activados, de forma automática, diversos mecanismos da framework, o que torna as aplicações mais simples de manter e compreender.

O Rails oferece suporte para AJAX e para construção de interfaces RESTful (Fielding, 2000), o que permite o desenvolvimento de aplicações Web 2.0 seguindo as melhores práticas e recomendações web (W3C, 2009a).

A framework Ruby on Rails foi extraída da construção de uma aplicação web concreta – Basecamp (37signals, 2009b), uma aplicação para gestão de projectos online da 37signals (37signals, 2009a) – o que lhe confere um carácter de framework para desenvolvimento de aplicações web.

A framework conta com uma comunidade alargada de utilizadores, o que potencia a sua evolução, assim como, o emergir do ecossistema que a envolve. A framework disponibiliza ainda um sistema de expansão por *plugins* e conta actualmente com um número significativo de *plugins* concebidos para diversos fins (Curtis, 2009).

A opção pela framework Ruby on Rails para o desenvolvimento da CrystalBox BaseStation

teve como base os seus princípios de desenho, a disponibilidade de documentação e de extensões oferecida pela sua extensa comunidade, assim como o seu carácter de framework para desenvolvimento de aplicações web.

5.5.2 CrystalBox BaseStation

A aplicação CrystalBox BaseStation é bastante simples, sendo apenas constituída por dois modelos de dados – `User` e `Project` – representando, respectivamente, os utilizadores do sistema e os seus projectos de documentação.

A aplicação utiliza controladores convencionais para a gestão de utilizadores e projectos, e para gestão de sessões de autenticação. Utiliza ainda um controlador dedicado para o acesso aos recursos dos projectos por parte dos clientes, com interface RESTful (Fielding, 2000) nos formatos HTML e JSON.

Configuração de projectos de documentação

A configuração de projectos de documentação é feita com recurso a um sistema flexível de configuração de propriedades por forma a possibilitar o registo de diferentes configurações sem necessidade de alterações ao nível dos modelos de dados ou dos controladores da aplicação.

Este nível de flexibilidade é fundamental para dar suporte à configuração de projectos de documentação para diferentes tipos de sistemas de controlo de versões, que requerem configurações distintas, assim como para diferentes tipos de recursos que podem exigir configurações próprias.

O sistema de configurações implementado tem como base o *plugin* Configurator (Dunn, 2009) que utiliza o sistema de associações polimórficas do Rails. O sistema possibilita a relação entre um determinado modelo e vários outros modelos numa única associação, recorrendo, para tal, não só à referência para o modelo (*foreign key*), mas também, ao tipo de modelo associado. A framework utiliza os poderosos mecanismos de introspecção do Ruby para estabelecer automaticamente as relações correctas para os modelos associados. Desta forma é possível criar configurações para qualquer tipo de modelo de forma extremamente flexível. A Figura 5.9 representa o diagrama de classes da associação polimórfica entre o modelo Configurator e outros modelos.

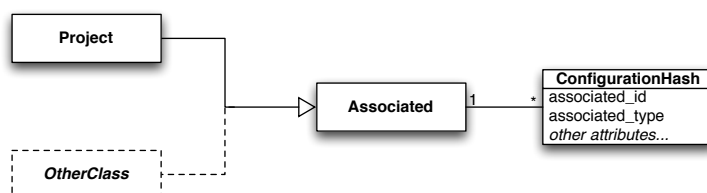


Figura 5.9: CrystalBox BaseStation – Diagrama de classes do sistema de configuração

Muito embora a aplicação apenas necessite actualmente de configurações ao nível dos projectos, não sendo para tal necessária a utilização de associações polimórficas, a utilização do *plugin*

Configurator facilitou o mecanismo de configurações, pelo que foi adoptado para o sistema de configurações da aplicação.

As configurações dos projectos de documentação são persistidas automaticamente pelo *plugin* Configurator no modelo `ConfiguratorHash`, e automaticamente referenciadas aos projectos, possibilitando um registo extremamente flexível de configurações. O *plugin* permite ainda a utilização de namespaces, o que permite uma melhor organização das configurações.

Desta forma, para suportar novas configurações dos projectos de documentação, é apenas necessário desenvolver novas vistas para edição (formulários) e para visualização (listagem) das mesmas, que apenas têm que definir novas chaves de configuração. Todo o suporte ao nível dos modelos e dos controladores está preparado para receber qualquer nova configuração.

Acesso a sistemas de controlo de versões

O acesso a sistemas de controlo de versões é suportado por uma hierarquia de classes que implementam uma interface comum para o acesso transparente aos recursos dos projectos.

A aplicação suporta o acesso directo a sistemas de controlo de versões Subversion, utilizando, para tal, uma biblioteca para interligação com a API do Subversion na linguagem Ruby, já disponível com a instalação do cliente Subversion. O acesso é garantido pela classe `Repository::Svn`, sendo o único tipo de acesso a suportar autenticação de utilizadores.

O suporte para outros sistemas de controlo de versões é, actualmente, feito por um processo simples de acesso a recursos livremente disponíveis na web por *browsers* de código como o ViewVC (ViewVC, 2009). Este tipo de acessos é suportado por sub-classes da classe `Repository::Web` que contém toda a lógica base para o acesso a este tipo de sistemas.

A Figura 5.10 representa um diagrama de classes dos sistemas de controlo de versões actualmente suportados pela aplicação.

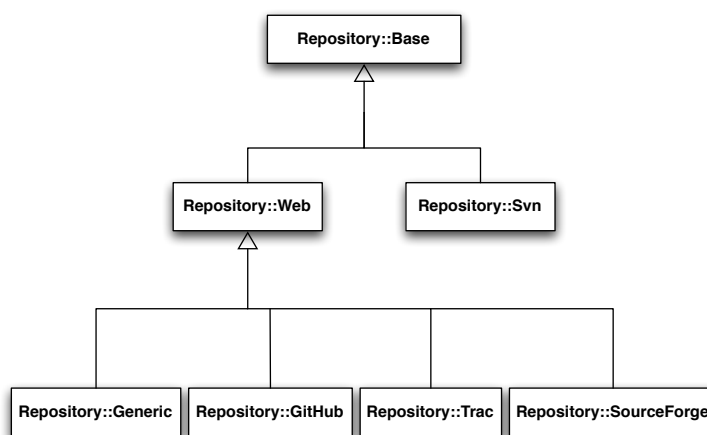


Figura 5.10: CrystalBox BaseStation – Diagrama de classes dos mecanismos de acesso a sistemas de controlo de versões da aplicação

O controlador da aplicação para acesso a recursos instancia dinamicamente uma das classes responsáveis pelo acesso ao sistemas de controlo de versões utilizado com base na configuração do projecto. O Ruby oferece um poderoso mecanismo de introspecção que facilita esse processo. A listagem abaixo representa o método `Project#get_repository` que instancia e retorna um objecto para acesso ao sistema de controlo de versões de um projecto com base nas suas configurações:

```
def get_repository
  "repository/#{config[:repository_type]}".
  classify.constantize.new(config)
end
```

A expansão da aplicação para suporte de novos acessos a sistemas de controlo de versões requer simplesmente a construção de uma nova subclasse de `Repository::Base` e novas vistas para a sua configuração nos projectos de documentação. A Secção 5.6.2 descreve com maior detalhe os mecanismos de evolução da CrystalBox BaseStation para o suporte de acesso a novos sistemas de controlo de versões.

Processamento de recursos

A aplicação disponibiliza uma outra hierarquia de classes para processamento de recursos. Esta hierarquia divide-se também em dois ramos principais: processamento de recursos do tipo texto (`Resource::Code`) e processamento de recursos binários (`Resource::Binary`). A aplicação dispõe actualmente de três sub-classes para processamento especializado de recursos do tipo texto: processamento de texto ou código-fonte genérico (`Resource::RawCode`), processamento de código-fonte Java (`Resource::Java`), e processamento de código-fonte Ruby (`Resource::Ruby`).

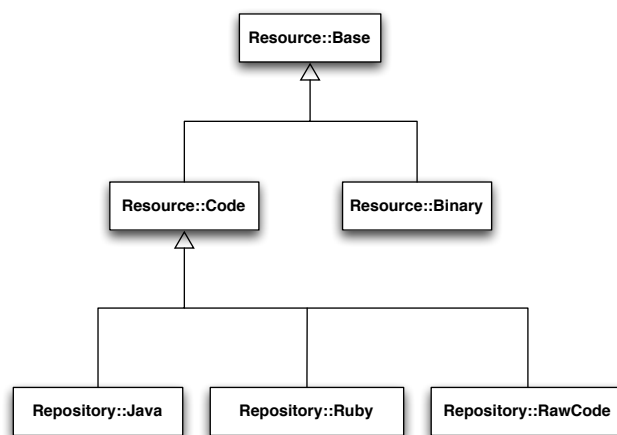


Figura 5.11: CrystalBox BaseStation – Diagrama de classes dos mecanismos de processamento de recursos da aplicação

Tal como no caso dos mecanismos de acesso a sistemas de controlo de versões, os mecanismos de processamento de recursos implementam uma interface comum e são instanciados dinamicamente pela aplicação com base nos parâmetros enviados pelo cliente para especificação dos recursos. A listagem abaixo representa o método de classe `Resource::Base#get_class_for` que é utilizado pelo controlador de acesso a recursos para instanciação do objecto responsável por processar o recurso pedido pelo cliente, com base nos parâmetros passados:

```
def self.get_class_for(params)
  if params.has_key?(:lang) # force a language
    begin
      "resource/#{params[:lang].downcase}".classify.constantize
    rescue NameError => e
      raise ResourceException.new(
        "<strong>#{params[:lang]}</strong> language is not supported!")
    end
  else
    # guess from mime type
    mime_type = MIME::Types.type_for(params[:path])[0]
    if mime_type and mime_type.binary?
      Resource::Binary
    else
      Resource::RawCode
    end
  end
end
```

A classe `Resource::Code` implementa um método para processamento sintáctico de código-fonte, que recorre à biblioteca externa Pygments (Pygments, 2009), e que está disponível para ser utilizado pelas suas sub-classes.

A classe `Resource::RawCode` utiliza simplesmente o mecanismo de processamento de formatação sintáctica disponibilizado pela sua super-classe (`Resource::code`) e permite cortar o texto por linhas através de um parâmetro específico para o efeito.

A classe `Resource::Java` utiliza ferramentas externas para processamento ao nível semântico de código-fonte Java – CrystalBox JavaTools, desenvolvidas especificamente para a CrystalBox, e utiliza também o mecanismo de formatação sintáctica disponibilizado pela sua super-classe (`Resource::code`).

A classe `Resource::Ruby` utiliza também uma ferramenta externa de processamento de código-fonte Ruby – CrystalBox RubyParser. O recurso a uma ferramenta externa na mesma linguagem da aplicação deve-se ao facto de esta recorrer dos mecanismos de interpretação de código-fonte Ruby existentes apenas na versão 1.9 do Ruby, versão ainda incompatível com a

framework Ruby on Rails que suporta a aplicação CrystalBox BaseStation. No futuro, esta ferramenta de processamento de código Ruby poderá ser incluída nativamente na aplicação. Esta classe utiliza igualmente o mecanismo de formatação sintáctica disponibilizado pela sua super-classe (`Resource::code`).

Ambas as ferramentas externas – CrystalBox JavaTools e CrystalBox RubyParser – são utilizadas pela aplicação CrystalBox BaseStation através da interface Ruby para execução de comandos no sistema operativo, o que permite a utilização da tecnologia que mais convier para cada caso concreto.

CrystalBox JavaTools CrystalBox JavaTools é um conjunto de ferramentas escritas na linguagem Java para processamento semântico de código-fonte Java.

As ferramentas CrystalBox JavaTools utilizam a biblioteca Eclipse AST para manipulação da árvore sintáctica abstracta (AST) de código Java, utilizada pelo próprio Eclipse IDE. A AST (Foundation, 2009a) é a framework base para muitas das poderosas ferramentas do Eclipse IDE (Foundation, 2009b), incluindo ferramentas de *refactoring*, Quick Fix e Quick Assist. A AST mapeia código-fonte Java numa representação em árvore, que constitui um formato mais conveniente e robusto para análise e modificação programática do código-fonte do que com base em texto.

A biblioteca Eclipse AST é utilizada pelas ferramentas CrystalBox JavaTools, para extracção de informação semântica do código, concretamente, para a localização de fragmentos de código: *classes, métodos e propriedades*.

A ferramenta disponibiliza uma interface Java nativa, e uma interface para acesso por comandos UNIX para que possa ser utilizada a partir da aplicação CrystalBox BaseStation.

Com base nestas ferramentas, a aplicação CrystalBox BaseStation é capaz de fornecer recursos aos clientes de documentação constituídos por fragmentos de código Java, extraídos de forma dinâmica e com base na semântica do código. Isto significa que, ainda que o código-fonte seja alterado, a aplicação é capaz de responder sempre com o mesmo recurso (fragmento de código), desde que a sua assinatura seja mantida.

CrystalBox RubyParser CrystalBox RubyParser é também uma ferramenta externa para interpretação da linguagem Ruby que utiliza o interpretador para a própria linguagem – Ripper – disponível na versão 1.9 do Ruby. O interpretador Ripper permite também criar representações do código-fonte Ruby em ASTs por forma a permitir a localização exacta de elementos do código-fonte, tais como módulos, classes e métodos. Com base na informação extraída da interpretação de código-fonte Ruby, a CrystalBox BaseStation permite, à semelhança do que acontece para a linguagem Java, fornecer fragmentos de código automaticamente extraídos do código-fonte, independentemente da localização que ocupam no código, ou da sua forma.

Cache

A aplicação CrystalBox BaseStation tem um sistema próprio para *cache* dos recursos servidos, para optimização de performance da resposta, uma vez que estes são provenientes de sistemas de

controlo de versões remotos e são, em alguns casos, sujeitos a processamentos consumidores de recursos temporais e computacionais significativos.

As ferramentas de processamento de recursos (nomeadamente para formatação sintáctica e processamento semântico de código-fonte) utilizam ficheiros temporários por serem ferramentas externas à aplicação. Uma vez que os recursos processados passam já por *filesystem*, o sistema de cache desenvolvido utiliza actualmente persistência também em *filesystem*, muito embora esta persistência possa vir a ser facilmente transferida no futuro para a base de dados, ou mesmo para memória, por forma a melhorar a performance na resposta.

A cache é feita apenas para recursos que indiquem uma revisão. Só nestes casos é possível identificar recursos imutáveis, passivos de serem persistidos em cache. Pela própria natureza dos sistemas de controlo de versões, os recursos que não indicam uma revisão, ou cuja revisão é a HEAD, são os recursos que acompanham a evolução do projecto, sofrendo naturalmente actualizações, pelo que a cache destes recursos não é efectuada. Sendo um dos objectivos da ferramenta oferecer a possibilidade de sincronismo de recursos conseguida pela actualização automática dos mesmos quando incluídos nas páginas de documentação, a cache de recursos da HEAD limitaria essa actualização, obrigando a uma intervenção para limpeza da cache por forma a serem disponibilizadas as versões mais actuais dos recursos.

Toda a lógica do mecanismo de cache está implementada na classe `Resource::Base`, e é utilizada, de forma transparente, por todas as suas sub-classes.

Interface RESTful

A aplicação disponibiliza uma interface RESTful que fornece acesso aos recursos dos projectos, por HTTP GET, nos formatos HTML e JSON. O formato JSON serve os recursos segundo a metodologia JSONP (metodologia apresentada mais à frente na Subsecção 5.5.3) para inclusão nas páginas de documentação por JavaScript, com recursos ao script `crystalbox.js`, e o formato HTML serve os recursos por acesso directo HTML no caso de o cliente ser incapaz de utilizar JavaScript.

A framework Ruby on Rails fornece mecanismos para criação de interfaces para diferentes formatos utilizando as mesmas acções e mesmos controladores.

A listagem abaixo representa o método do controlador dos recursos que faz a visualização dos erros encontrados no acesso aos recursos dos projectos de documentação.

```
def render_error(exception)
  @error = exception.message
  respond_to do |format|
    format.json do
      json_data = { :content =>
                    render_to_string(:template =>
                                     'resources/error.html.erb') }
      render :json => json_data.to_json,
```

```

        :callback => params[:jsoncallback]
    end
    format.html do
        render :template => 'resources/error',
              :layout => 'resource'
    end
end
end
end

```

5.5.3 CrystalBox Client

O *script* `crystalbox.js` é um ficheiro de código JavaScript que contém toda a lógica da camada de cliente da ferramenta CrystalBox. Este *script* está disponível online⁴ para que possa ser directamente incluído no código HTML que chega ao cliente. A listagem seguinte contém o código HTML necessário para incluir o *script* numa página de documentação.

```

<script type="text/javascript"
    src="http://crystalbox.org/crystalbox.js"></script>

```

Uma vez disponível na página, este script é responsável por todo o processamento das referências a recursos CrystalBox.

A utilização do *script* `crystalbox.js` tem como objectivo aceder e disponibilizar recursos de forma assíncrona nas páginas de documentação, utilizando para tal, tecnologia JavaScript. Os recursos são incluídos no DOM da página HTML por JavaScript, mecanismo fundamental para inserir recursos que não estão ainda disponíveis na página que chega ao cliente proveniente do servidor web, geralmente um wiki.

XMLHttpRequest

O objecto XMLHttpRequest, disponível na generalidade dos *web browsers* actuais, implementa uma interface, exposta por um motor de *scripting*, que permite que *scripts* efectuem funcionalidades de cliente HTTP, tais como submissões de formulários, ou o carregamento de dados de um servidor (W3C, 2009b).

O objecto tem como nome XMLHttpRequest por retro compatibilidade com a web, apesar de cada componente do nome ser potencialmente enganadora. Em primeiro lugar, o objecto suporta qualquer formato baseado em texto, e não apenas XML. Segundo, o objecto pode ser usado para fazer pedidos tanto sobre HTTP, como HTTPS. Finalmente, o objecto suporta todas as actividades envolvidas com as solicitações HTTP ou respostas para os métodos definidos no protocolo HTTP, e não apenas pedidos, como o nome sugere.

⁴Disponível em <http://crystalbox.org/crystalbox.js>

Este objecto é normalmente utilizado para comunicar de forma assíncrona e transparente com o servidor para enriquecimento da interacção com as aplicações web, tornando-se à partida apelativo para a sua utilização no acesso a recursos disponibilizados pela aplicação CrystalBox BaseStation.

Contudo, a utilização do objecto XMLHttpRequest fica confinada apenas à ligação ao servidor que serve a página base por uma limitação de segurança dos browsers conhecida por *Same Origin Policy*. Esta medida de segurança existe para evitar ataques por *cross site scripting* (XSS) a que os clientes ficariam sujeitos caso o objecto XMLHttpRequest pudesse fazer acesso a outros servidores.

A Figura 5.12 representa a tentativa de acesso a recursos disponíveis na aplicação CrystalBox BaseStation utilizando o objecto XMLHttpRequest que é inviabilizado pela restrição de segurança *Same Origin Policy*.

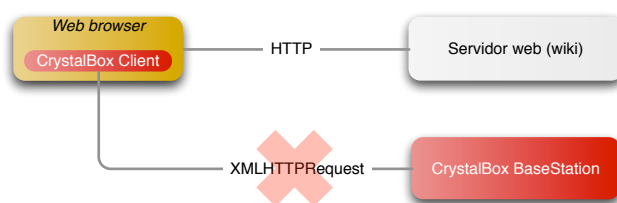


Figura 5.12: Tentativa de acesso a recursos disponíveis na aplicação CrystalBox BaseStation utilizando o objecto XMLHttpRequest

JSONRequest

A especificação JSONRequest (Crockford, 2006) surgiu como uma proposta para ultrapassar as limitações impostas pela restrição de segurança *Same Origin Policy*, por forma a permitir o acesso a dados provenientes de outros servidores, sem comprometer a segurança do cliente.

A especificação propõe um serviço de troca de dados seguro e fiável permitindo que um *script* se ligue, a partir de uma página, a qualquer servidor externo para efectuar trocas de dados.

O JSON (JSON, 2009) é um formato de transmissão de dados baseado num subconjunto seguro do JavaScript. O JSON permite a representação de dados estruturados, simples ou complexos, não sendo possível a representação de funções ou expressões. JSON é estritamente uma representação de dados com regras específicas e precisas de sintaxe, sendo, por isso, muito simples de determinar se um texto JSON é sintacticamente correto. Um objecto JSON serializado em forma de texto pode facilmente ser convertido num valor JavaScript, o que o torna num modelo conveniente para utilização nesta linguagem, muito embora exista suporte para a sua utilização em diversas outras linguagens como Ruby, Python, PHP, Perl, Java, etc.

O objecto JSONRequest proposto por Douglas Crockford disponibiliza três métodos: *post*, *get* e *cancel*. A especificação garante que a troca de dados com os servidores é feita exclusivamente no formato JSON, não permitindo o envio de funções ou expressões que ponham em causa a segurança dos intervenientes. Este facto, aliado a uma outra série de restrições de segurança que a especificação prevê, torna-a suficientemente segura para que venha a ser implementada nos

browsers isenta da restrição de segurança *Same Origin Policy*, abrindo portas para o acesso a APIs por JavaScript, potenciando ainda mais a web orientada aos serviços (SOA).

A especificação XMLHttpRequest prevê ainda suporte para comunicação bidireccional com o servidor, o que permite que o servidor inicialize transmissões assíncronas, potenciando a sua utilização para notificações em tempo real, ou aplicações colaborativas, tais como aplicações de *instant messaging*.

Muito embora o objecto XMLHttpRequest venha possivelmente a ser adoptado pelos *web browsers* para acesso a servidores externos por JavaScript, utilizando JSON como formato de transmissão de dados, não existem ainda implementações disponíveis nos *web browsers* actuais, pelo que, a solução XMLHttpRequest não constitui ainda uma resposta concreta ao problema.

A Figura 5.13 representa um hipotético acesso a recursos disponíveis na aplicação CrystalBox BaseStation utilizando o objecto XMLHttpRequest que permitiria ultrapassar os constrangimentos de segurança impostos ao objecto XMLHttpRequest.

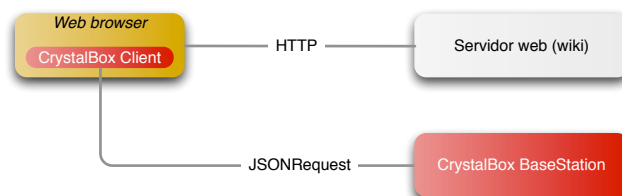


Figura 5.13: Hipotética utilização do objecto XMLHttpRequest para acesso a recursos disponíveis na aplicação CrystalBox BaseStation

Soluções tradicionais

De entre as soluções tradicionais para ultrapassar o problema de acesso a dados provenientes de servidores externos de forma assíncrona, destacam-se as seguintes:

Proxy local Instalação no servidor de origem de um *proxy* que recebe os pedidos por XMLHttpRequest e os processa com o servidor externo. Esta solução não seria viável para a ferramenta CrystalBox uma vez que obrigaria à instalação de um proxy no servidor de origem das páginas de documentação (tipicamente um wiki), o que vai totalmente contra o objectivo deste trabalho, uma vez que obrigaria a intervenções nos servidores de documentação.

Flash A utilização de tecnologia Flash permite o acesso assíncrono a servidores externos, desde que seja disponibilizado no servidor de origem da página um ficheiro `crossdomain.xml` com a lista de servidores externos acessíveis, o que colide, mais uma vez, com o objectivo deste trabalho, uma vez que obrigaria também a intervenções no servidor de origem.

Script tag A restrição de segurança *Same Origin Policy* pode ser contornada pela inclusão de *script tags* com URLs fonte de domínios externos. Os scripts carregados desta forma são executados localmente sem qualquer restrição de segurança, permitindo explorar o acesso

assíncrono por JavaScript a servidores externos. No entanto, é difícil saber a altura em que o conteúdo está disponível na página e, não existindo nenhuma metodologia padrão, pode ser considerado um “risco de segurança”.

JSONP

O JSONP (Ippolito, 2005) – JSON with Padding – surgiu como uma proposta de normalização para permitir o acesso a dados no formato JSON, provenientes de domínios externos, utilizando *script tags*.

O autor, Bob Ippolito, pretende tornar os scripts remotos mais flexíveis, para que um cliente possa controlar o modo como eles funcionam. Nos termos da proposta, o script local simplesmente indica ao script remoto um nome que este deverá colocar no início da resposta seguido dos dados JSON envolvidos entre parêntesis. Desta forma, esse nome funcionará como uma chamada a uma função, que deverá estar definida no cliente, por forma a funcionar como *callback*, quando a resposta chega ao cliente e é executada em JavaScript. O cliente poderá assim utilizar os serviços que implementem esta metodologia da forma que entender.

A listagem abaixo representa a resposta obtida da API JSON do serviço web de *bookmarking* Delicious (Delicious, 2009) quando acedida através do seguinte URL:

```
http://feeds.delicious.com/v2/json/nunobaldaia/
jsonp+padding?callback=callback_function_name

callback_function_name([ {
  "u": "http://bob.pythonmac.org/archives/2005/12/05/...",
  "d": "from __future__ import * \u00bb Remote JSON - JSONP",
  "t": ["jsonp", "json", "padding", "callback"],
  "dt": "2009-07-21T22:50:32Z",
  "n": "",
  "a": "nunobaldaia"
} ])
```

Neste exemplo, os dados JSON do resultado da chamada à API foram envolvidos entre parêntesis e precedidos do parâmetro passado pela variável *callback* (*callback_function_name*). O script do cliente que fez este pedido deverá definir uma função com o mesmo nome (*callback_function_name*) que será invocada quando o resultado estiver disponível, uma vez que este é executado como JavaScript, funcionando como *callback* do pedido.

Esta metodologia é hoje globalmente adoptada para acesso a APIs e Web Services de plataformas web como o Delicious, Flickr, Google, Twitter, Yahoo!, YouTube, etc, (Delicious, 2009; Flickr, 2009; Google, 2009a; Twitter, 2009; Inc., 2009a; YouTube, 2009) em formato JSON.

A aplicação CrystalBox BaseStation disponibiliza uma API REST que responde em formato JSON, permitindo o acesso, por JSONP, aos recursos por ela disponibilizados, a partir de páginas de documentação provenientes de qualquer outro servidor de origem. A figura 5.14 ilustra a

solução adoptada pela ferramenta CrystalBox para acesso assíncrono por JavaScript aos recursos provenientes da aplicação CrystalBox BaseStation.

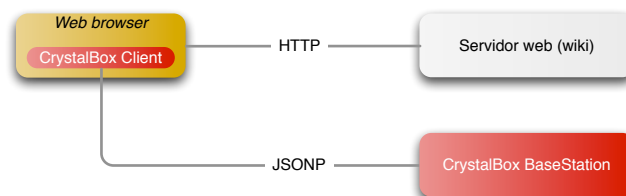


Figura 5.14: Acesso a recursos da CrystalBox BaseStation por JSONP

jQuery

Com o emergir da era Web 2.0 foram surgindo diversas frameworks JavaScript *open-source* que facilitam o desenvolvimento de aplicações baseadas em JavaScript, nomeadamente, jQuery, Prototype, MooTools, Yahoo! UI Library (YUI), entre várias outras (Resig, 2009; Prototype, 2009; Proietti, 2009; Inc., 2009b). De forma geral, todas elas oferecem o mesmo tipo de funcionalidades, tais como, selecção e manipulação de elementos DOM, eventos, manipulação de CSS, efeitos e animações, Ajax e utilidades JavaScript.

jQuery (Resig, 2009) é uma biblioteca JavaScript *open-source* bastante leve que, num curto espaço de tempo se tornou uma das mais populares bibliotecas na web. A framework permite, de forma elegante e eficaz, encontrar, navegar e manipular elementos do DOM através de um selector de elementos bastante simples e poderoso baseado em CSS3 e XPath. Os comandos jQuery podem ser encadeados em sequência e podem operar sobre conjuntos de elementos de forma simples e transparente, o que permite escrita de código conciso e fácil de compreender. A jQuery inclui também um conjunto de APIs para Ajax, efeitos e animações, manipulação de CSS, utilidades e um mecanismo bastante simples de extensibilidade por meio de plugins, contando já com uma oferta alargada de plugins externos.

Muito embora de forma geral todas as frameworks existentes no mercado ofereçam interfaces transparentes para acesso assíncrono a servidores externos, utilizando o mecanismo JSONP, a biblioteca jQuery foi adoptada para a ferramenta CrystalBox, uma vez ser bastante leve, rápida e simples de utilizar.

Script `crystalbox.js`

O *script* `crystalbox.js` inclui, num mesmo ficheiro, o código da biblioteca jQuery, o código do plugin jQuery FaceBox para abertura de recursos nas caixas CrystalBox, e, por fim, o código dedicado ao processamento de recursos CrystalBox.

A inclusão de todo o código num só ficheiro tem como objectivo, por um lado, facilitar a utilização da ferramenta CrystalBox pela inclusão de apenas uma *tag script* no HTML da página de

documentação e, por outro, minimizar o número de pedidos ao servidor por questões de performance, uma vez que os browsers limitam geralmente a dois, o número de pedidos em simultâneo para cada domínio.

O *script* `crystalbox.js` faz ainda a inclusão automática do link para a folha de estilos `crystalbox.css`⁵ no DOM da página do cliente. Assim, os únicos ficheiros descarregados para o cliente são o *script* `crystalbox.js` e a folha de estilos `crystalbox.css`.

O *script* faz a configuração da biblioteca jQuery para que seja utilizada sem conflitos, ou seja, sem a utilização da variável `$`, típica das bibliotecas JavaScript, por forma a poder ser utilizada em páginas de documentação que possam eventualmente utilizar outras bibliotecas.

5.6 Mecanismos de evolução

A aplicação CrystalBox Base Station foi desenhada por forma a permitir uma fácil evolução no que diz respeito ao suporte para novos recursos, como por exemplo, suporte para novas linguagens, assim como ao suporte para acesso a novos sistemas de controlo de versões.

5.6.1 Suporte para novos recursos

O suporte para novos recursos é possível por extensão da classe `Resource::Base`, existente na pasta `lib/` da aplicação, base para qualquer tipo de recurso suportado pela aplicação CrystalBox BaseStation. Esta classe base disponibiliza mecanismos comuns de inicialização e de gestão de cache, bem como um método de classe para inferência automática da sub-classe representativa de um determinado recurso com base nos seus parâmetros.

A classe base `Resource::Base` tem duas subclasses principais para suporte de *recursos do tipo texto* e *recursos binários*, representadas respectivamente pelas classes `Resource::Code` e `Resource::Binary`.

A Figura 5.11 representa o diagrama da hierarquia de classes representativas de recursos, actualmente disponíveis na aplicação.

Recursos do tipo texto

Os recursos do tipo texto são suportados por extensões à sub-classe base `Resource::Code` que disponibiliza um mecanismo comum para formatação sintáctica de código, ficando o tratamento ao nível semântico dos diferentes tipos de recursos a cargo das sub-classes específicas.

As sub-classes de um recurso do tipo texto (extensões da sub-classe `Resource::Code`) devem disponibilizar o método público `get_processed_code` que deverá retornar o código já processado, tanto ao nível semântico como ao nível sintáctico, devendo, para tal, fazer uso dos mecanismos de formatação sintáctica oferecidos pela classe base `Resource::Code`.

As sub-classes específicas para recursos de código-fonte nas linguagens Java e Ruby – `Resource::Java` e `Resource::Ruby` respectivamente – são exemplos de representações de

⁵Disponível em <http://crystalbox.org/crystalbox.css>

recursos do tipo texto, que fazem uso de ferramentas externas para processamento semântico de código-fonte, e do mecanismo comum de formatação sintáctica oferecido pela sub-classe base – `Resource::Code`.

Recursos binários

Os recursos do tipo binário são processados pela sub-classe `Resource::Binary` que disponibiliza um único método público – `get_binary_data` – que retorna os dados do recurso binário para que possam ser enviados pelo controlador directamente para o cliente.

Ao nível do controlador, os recursos binários são tratados de forma distinta dos recursos do tipo texto. Para este tipo de recursos, o controlador responde no formato JSON com uma *tag* HTML que referencia novamente o mesmo recurso, mas desta vez no formato HTML para que seja servido directamente ao cliente com o MIME-Type correspondente, que é automaticamente inferido a partir da extensão do nome do ficheiro do recurso. Os recursos binários são, por tanto, servidos em duas fases: 1) código HTML com referência para o recurso binário; 2) envio dos dados do recurso no MIME-Type correspondente.

5.6.2 Suporte para acesso a novos sistemas de controlo de versões

O suporte para acesso a novos sistemas de controlo de versões (VCS) é possível por extensão da classe `Repository::Base`, existente na pasta `lib/` da aplicação.

A Figura 5.10 representa o diagrama da hierarquia de classes para acesso a VCSs actualmente suportados pela aplicação. A sub-classe `Repository::Svn` suporta o acesso nativo a sistemas Subversion. Os restantes acessos actualmente disponíveis são suportados por extensões da sub-classe base `Repository::Web` que permitem o acesso a recursos disponíveis em sistemas de controlo de versões abertos e acessíveis por HTTP segundo um padrão de URLs.

As sub-classes de extensão à classe base `Resource::Base` podem sobrepor o método `initialize` da classe base por forma a permitir a inicialização específica dos objectos de cada sub-classe, devendo garantir também a inicialização definida no método da classe base através da invocação do método `super`. Para além da correcta inicialização, as sub-classes devem disponibilizar o método `cat` para extracção de recursos para uma determinada *path* e revisão, e o método `unique_key` que deverá retornar uma chave única para um determinado recurso numa determinada revisão que será utilizado como chave única para a *cache* do recurso.

O acesso a sistemas de controlo de versões abertos e acessíveis por HTTP é suportado pela classe `Repository::Web`, ficando a cargo das suas sub-classes apenas a inicialização da expressão regular do padrão de URL para acesso a recursos específico de cada sistema.

O acesso nativo a outros sistemas de controlo de versões deverá ser feito por extensão da classe base `Resource::Base` e deverá implementar, pelo menos, os métodos `cat` e `unique_key`.

5.7 Considerações finais

No contexto dos objectivos delineados para este trabalho, a ferramenta CrystalBox resolveu de forma eficaz o principal problema encontrado no XSDoc que se resume à sua dificuldade de integração noutros wikis. Este problema foi experimentado nos projectos exploratórios efectuados para a adaptação do XSDoc aos wikis SnipSnap e MoinMoin tendo fornecido uma percepção valiosa da sua considerável dimensão e complexidade.

No entanto, o XSDoc não é apenas uma ferramenta de documentação de software, mas sim todo um sistema de apoio à documentação de frameworks que a ferramenta CrystalBox não tenciona replicar. O resultado final da CrystalBox é, pelo contrário, um mecanismo leve para integração em sistemas de edição web, entre os quais se incluem os wikis, por forma a permitir a transclusão de artefactos de software garantindo a preservação da sua consistência semântica, para a produção de documentação em sintonia com os projectos de desenvolvimento. Desta forma, é possível integrar este interessante e importante conceito, originário do XSDoc, em virtualmente qualquer sistema de edição de documentação que for mais conveniente para cada projecto em particular.

Capítulo 6

Validação

A validação do conceito materializado pela ferramenta de apoio à documentação CrystalBox foi feita com recurso a três casos de estudo:

1. a documentação da própria aplicação CrystalBox BaseStation;
2. um caso de estudo real em contexto académico;
3. um caso de estudo real em ambiente empresarial.

6.1 Caso de estudo 1 – CrystalBox BaseStation

A ferramenta desenvolvida foi utilizada para documentar a sua forma de extensão para suporte de novos sistemas de controlo de versões, para suporte a novos processadores de recursos, assim como para a criação de novas macros para outros wikis.

Para o desenvolvimento do projecto CrystalBox foi utilizado o sistema de controlo de versões Git (Git, 2009). Embora, o projecto não esteja disponível em *open-source* no GitHub (Awesome, 2009), único sistema de controlo de versões para Git suportado pela CrystalBox, a flexibilidade oferecida pela ferramenta CrystalBox para configuração genérica de sistemas de controlo de versões acessíveis pela web possibilitou a configuração do próprio projecto para documentação de forma relativamente fácil. O sistema Git inclui o Gitweb (Sievers, 2009), uma interface web para exploração de repositórios locais, o que permite configurar um projecto de documentação utilizando uma configuração genérica. A Figura 6.1 representa a configuração do projecto de documentação da própria CrystalBox numa configuração local, utilizando o Gitweb.

A documentação foi produzida no wiki DokuWiki (DokuWiki, 2009) na forma de um *cookbook* (Aguiar, 2003) (Pág. 75), com recurso à macro CrystalBox disponibilizada. O *cookbook* inclui três receitas (*recipes*):

1. receita para extensão da aplicação CrystalBox BaseStation para suporte de novos sistemas de controlo de versões;

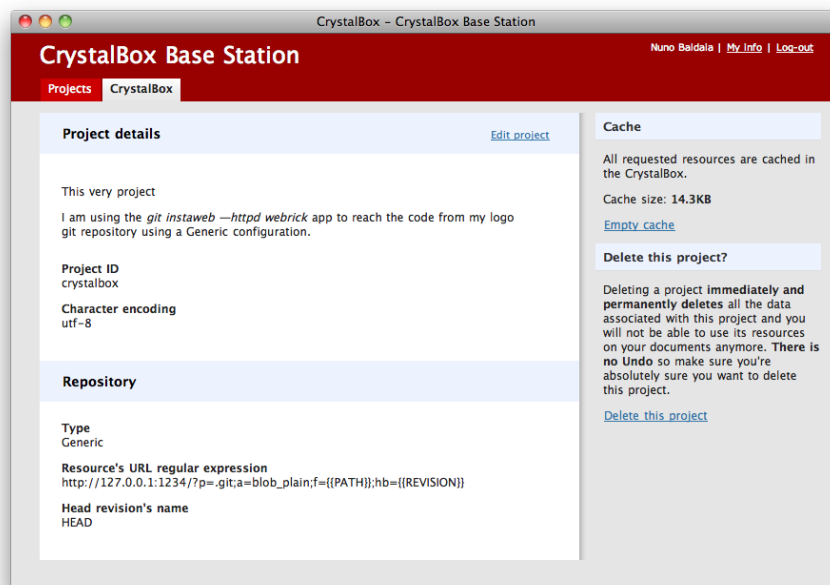


Figura 6.1: Configuração do projecto para documentação da CrystalBox

2. receita para extensão da aplicação CrystalBox BaseStation para suporte de novos processadores de recursos para suporte de novas linguagens, ou outro tipo de processamento, como por exemplo para geração automática de diagramas UML a partir de código-fonte;
3. receita para construção de novas macros para facilitar a utilização da ferramenta CrystalBox a partir de outros wikis, ou outros sistemas de autoria web.

A possibilidade de inclusão de fragmentos de código síncronos com o projecto de desenvolvimento por simples referência de recursos demonstrou ser verdadeiramente eficaz uma vez que possibilita a escrita prematura de documentação, ainda na fase de desenvolvimento, sem que se perca a validade da documentação produzida com o evoluir do projecto. A Figura 6.2 representa um exemplo da evolução de um recurso referenciado que é alterado no tempo. No exemplo é utilizado um fragmento de código-fonte (método) que é referenciado dinamicamente em alturas diferentes da evolução do projecto. O sistema encarrega-se de incluir automaticamente o código síncrono com a evolução do projecto, não havendo necessidade de qualquer intervenção na documentação produzida.

A facilidade de referência de recursos oferecida pelas macros dedicadas é também um forte incentivo à utilização de recursos provenientes do projecto de desenvolvimento, e consequentemente ao enriquecimento da documentação produzida. A listagem seguinte contém a *markup* utilizada no DokuWiki para a inclusão do fragmento de código da Figura 6.2 representada no exemplo anterior.

```
<crystalbox path=
  "crystalbox/crystalbox_tools/ruby/parse_ruby.rb?member=on_class">
```



Figura 6.2: Evolução de um recurso CrystalBox referenciado ao longo do tempo

```
crystalbox_tools/ruby/parse_ruby.rb#on_class
</crystalbox>
```

A produção do *cookbook* foi um bom caso de estudo para a validação da ferramenta desenvolvida uma vez que a utilização de referências directas a fragmentos de código é de extrema importância para documentar e exemplificar as extensões possíveis ao projecto.

A ligação a recursos na caixa CrystalBox permite a visualização e carregamento de recursos apenas quando o leitor o pretender, sem ocupar espaço de formatação na documentação produzida. Para os casos em que se pretende realmente evidenciar um fragmento de código que se está a descrever, ou para que este esteja presente na documentação impressa, a inclusão de recursos *inline* é também fundamental. A Figura 6.3 representa a documentação produzida para uma receita do

cookbook, na qual são utilizadas tanto referências inline, como referências normais para recursos do projecto.

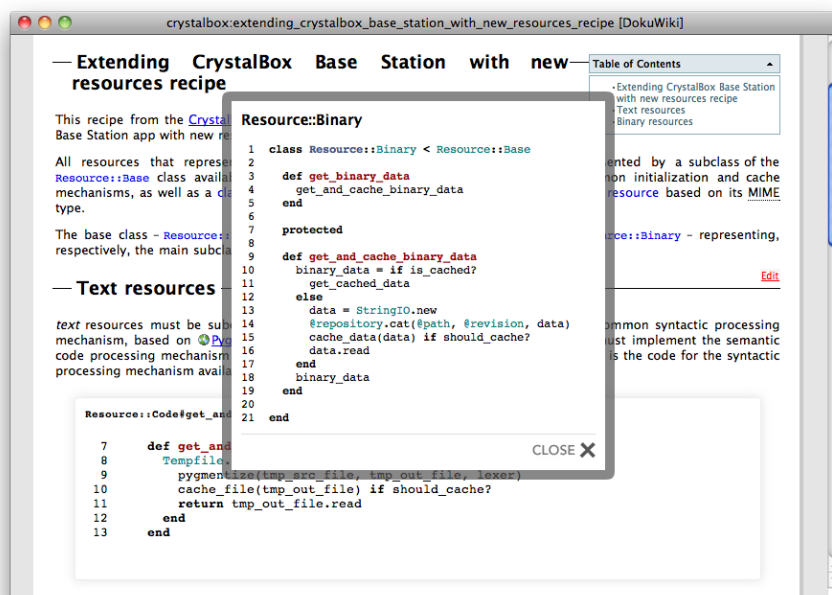


Figura 6.3: Documentação produzida com a CrystalBox

6.2 Caso de estudo 2 – Disciplina MIEIC/LDSO-2009/10

A ferramenta CrystalBox está actualmente a ser utilizada em contexto académico para a produção de documentação de projectos em desenvolvimento na unidade curricular de Laboratório de Desenvolvimento de Software (LDSO) do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) da Faculdade de Engenharia da Universidade do Porto (FEUP).

A ferramenta de documentação utilizada foi também o wiki DokuWiki (DokuWiki, 2009) integrado com a ferramenta CrystalBox através da macro disponibilizada para o efeito.

A figura 6.4 ilustra um exemplo da utilização da ferramenta CrystalBox para auxílio na produção de documentação de um projecto em Ruby no wiki DokuWiki.

6.3 Caso de estudo 3 – Plataforma web escolinhas.pt

O Escolinhas é uma plataforma web¹, colaborativa e social para Escolas EB1/2.

“As Escolinhas são espaços educativos de colaboração, comunicação e partilha entre alunos, pais e professores do Ensino Básico, acessíveis em www.escolinhas.pt.

¹Disponível em <http://www.escolinhas.pt/>

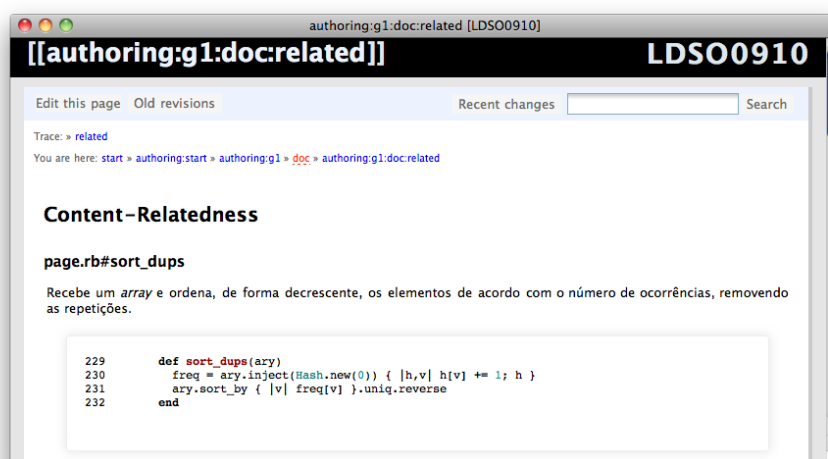


Figura 6.4: Documentação com recurso à CrystalBox na disciplina MIEIC/LDSO-2009/10

De uma forma muito simples e natural, as Escolinhas promovem a utilização efectiva das Tecnologias de Informação e Comunicação (computadores, Internet e quadros interactivos), nas práticas de ensino formal, em contexto de aula ou extra curricular, dentro e fora do recinto escolar.

Fortemente ancorada na estrutura social e organizativa da escola física, as Escolinhas permitem a livre criação de textos e desenhos por alunos dos 4 aos 12 anos, bem como a sua partilha pela comunidade escolar (alunos, pais, professores).

Nas Escolinhas é possível ler, escrever, pintar, desenhar, brincar, colaborar, partilhar e estar com os colegas de escola, amigos, pais e professores.” (Escolinhas, 2009)

A plataforma Escolinhas é desenvolvida num ambiente empresarial, concretamente na empresa Tecla Colorida, um *spin-off* da FEUP e INESC Porto.

Do ponto de vista tecnológico, a plataforma Escolinhas é uma aplicação web desenvolvida sobre a framework de desenvolvimento web Ruby on Rails (Hansson, 2009) e segue uma metodologia ágil de desenvolvimento.

A CrystalBox tem vindo a ser utilizada para apoio à documentação de conhecimento de alto nível da aplicação, nomeadamente para pormenores de arquitectura e soluções mais estruturantes e de extensão da aplicação. Outra actividade de documentação para a qual a ferramenta se tem demonstrado útil é no apoio à escrita de documentos técnicos que focam um contexto de uma determinada funcionalidade, para os quais a possibilidade de inclusão de fragmentos de código, por vezes provenientes de diversos ficheiros. A plataforma base para documentação da aplicação é o DokuWiki (DokuWiki, 2009) e a CrystalBox é integrada no wiki, mais uma vez, utilizando a macro disponibilizada. A Figura 6.5 representa uma página de documentação do projecto escolinhas no wiki de apoio ao projecto.

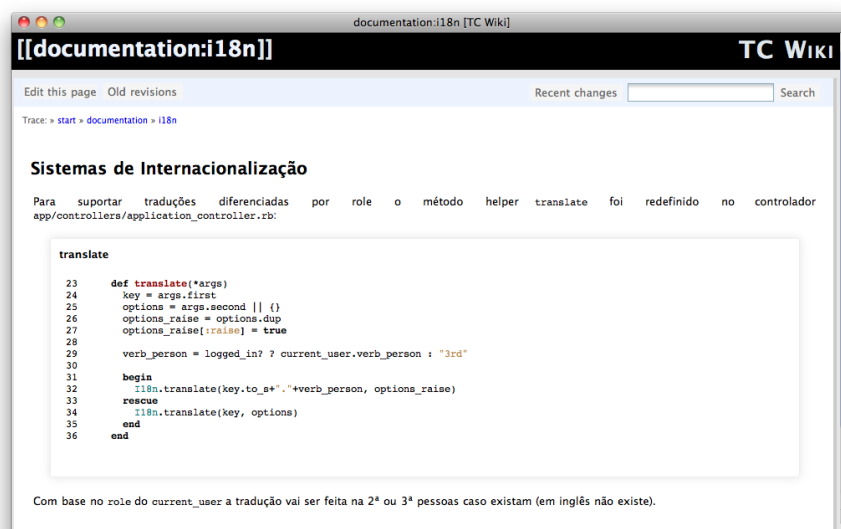


Figura 6.5: Documentação com recurso à CrystalBox da plataforma web escolinhas.pt

6.4 Considerações finais

Embora não tenha sido possível fazer um estudo empírico para validação desta tese, estes três casos de estudo forneceram já uma percepção real das vantagens da utilização da ferramenta CrystalBox, que tem sido natural e espontaneamente utilizada para a transclusão de artefactos na produção de documentação e planeamento de projectos de software.

A facilidade oferecida pela ferramenta para a referência de artefactos externos tem-se revelado num forte incentivo à utilização, por exemplo, da inclusão de fragmentos de código-fonte nos documentos produzidos, uma vez que, para tal, basta criar uma referência através de uma simples linguagem de *markup*.

A facilidade de adopção e utilização da ferramenta ficou comprovada pelo facto de, em poucos minutos, várias equipa de desenvolvimento de software terem começado a utilizar a ferramenta CrystalBox para a produção de documentação com base num wiki, para o qual existe uma macro disponibilizada.

Capítulo 7

Conclusões e trabalho futuro

A ferramenta de documentação de software CrystalBox demonstrou cumprir com sucesso os principais objectivos delineados para este trabalho. O conceito explorado demonstrou ter bastante potencial e a ferramenta desenvolvida constitui uma boa base de evolução futura.

A opção pela utilização de repositórios de código em sistemas de controlo de versões para acesso síncrono a artefactos do projecto de desenvolvimento permite a produção de documentação externa segundo os métodos *multiple-source*, garantindo a consistência semântica entre documentação e artefactos, através de mecanismos de referênciação, em alternativa ao tradicional mecanismo de *copy/paste*.

A opção pela centralização dos mecanismos de acesso e processamento de artefactos na aplicação web CrystalBox BaseStation e a disponibilização da leve camada de cliente CrystalBox Client para integração simplificada em ferramentas de produção de documentação, permite uma livre escolha do suporte de documentação. Este suporte poderá, desta forma, ser qualquer ferramenta de edição de HTML, muito embora, a utilização de wikis tenha, para além das vantagens inerentes a estes sistemas, a possibilidade de utilização de extensões (*macros*) que facilitem o processo de referênciação de recursos. Esta facilidade de integração em qualquer ferramenta de edição web, foi um dos principais objectivos deste trabalho, delineado no sentido de potenciar a adopção da ferramenta em diferentes contextos.

A aplicação CrystalBox BaseStation foi desenvolvida segundo uma arquitectura que permite o suporte e fácil extensão para processamento de diversos tipos de recursos (ex. processamento de código-fonte em diferentes linguagens de programação), assim como para conexão a diversos sistemas de controlo de versões para acesso a recursos provenientes de repositórios remotos de projectos de desenvolvimento de software. Este carácter de extensibilidade possibilita a utilização da ferramenta CrystalBox para documentação de projectos em diferentes contextos e realidades.

Relativamente às técnicas de documentação de software existentes, abordadas na Secção 2.1, a ferramenta CrystalBox constitui, no o âmbito deste trabalho, um avanço significativo no que diz respeito à flexibilidade, facilidade de adopção e utilização para produção de documentação de software.

7.1 Problemas identificados

Muito embora o objectivo principal da ferramenta de documentação CrystalBox seja suportar a documentação ágil de software em ambientes de desenvolvimento *open-source*, a ferramenta pode ser utilizada para a documentação de software em geral. No entanto, para projectos fechados, surgem dois problemas: 1) a necessidade da cedência das credenciais para acesso ao sistema de controlo de versões à aplicação CrystalBox BaseStation; 2) a dificuldade de protecção do acesso aos recursos provenientes do sistema de controlo de versões.

O acesso a sistemas de controlo de versões fechados, actualmente suportado para sistemas Subversion, obriga à cedência das credenciais para acesso ao sistema de controlo de versões à aplicação CrystalBox BaseStation. Esta cedência é necessária para que a aplicação possa fazer o acesso aos recursos disponíveis no sistema de controlo de versões mas pode ser dissuasora para projectos com algum nível de segurança no que diz respeito à protecção da informação e código. No entanto, embora num cenário um pouco afastado do objectivo principal da CrystalBox, a ferramenta poderá ser instalada num servidor da própria instituição, resolvendo este potencial problema.

Adicionalmente, o acesso aos recursos por parte da camada de cliente CrystaBox Client é feito directamente por acesso à API RESTful da aplicação CrystalBox BaseStation que não suporta qualquer sistema de autenticação. Esta falta de sistemas de protecção de acesso deve-se ao conceito e arquitectura geral da ferramenta, na qual o acesso é feito directamente a partir do cliente por JavaScript sem a necessidade de qualquer outro tipo de dependências tecnológicas, o que dificulta a concepção de um sistema seguro.

Estes são os dois grandes problemas identificados mas que não têm grande efeito para a documentação de projectos *open-source*, que são o principal alvo do objectivo deste trabalho.

7.2 Trabalho futuro

A extensão da ferramenta CrystalBox BaseStation para processamento semântico de outras linguagens, por recurso a novas ferramentas de processamento, semelhantes às ferramentas de processamento para as linguagens Java e Ruby, será um dos passos a dar para aumentar significativamente a utilidade e, consequentemente a adopção desta ferramenta em grande escala.

A navegabilidade do código semanticamente enriquecido seria uma evolução muito significativa para esta ferramenta de documentação, uma vez que permitiria a exploração de um projecto inteiro, com início num fragmento de código, dentro de uma caixa CrystalBox.

O suporte para outras representações de recursos, como por exemplo a geração automática de diagramas UML por engenharia reversa de código fonte, seria uma funcionalidade exequível e com muito valor para a documentação de software.

A protecção do acesso aos recursos fornecidos pela aplicação CrystalBox BaseStation por um sistema seguro de credenciais de autenticação, permitiria a utilização da ferramenta em projectos privados.

A possibilidade de escrita livre de código no próprio wiki que seria formatado e apresentado da mesma forma que são os recursos provenientes dos repositórios de código, seria uma funcionalidade relativamente simples de implementar e que traria uma maior agilidade à ferramenta de documentação. Esta funcionalidade permitiria escrever exemplos de utilização de código que não existam nos projectos de desenvolvimento, utilizando o mesmo formato de apresentação.

O suporte nativo para acesso a outros sistemas de controlo de versões, como por exemplo o CVS, o Mercurial, ou o Git, permitiria o acesso a mais repositórios de projectos privados, o que abrangeria ainda mais o possível espectro de utilização desta ferramenta em ambiente empresarial.

A integração de editores de documentação em IDEs potenciaria ainda mais a experiência de documentação uma vez que permitia desenvolver e documentar projectos de software num mesmo ambiente comum, eliminando a necessidade da troca constante entre os ambientes de desenvolvimento e de documentação, o que seria mais um incentivo à documentação em simultâneo com o desenho e desenvolvimento de software. Por outro lado, a integração de uma ferramenta de documentação em IDEs poderia tirar partido dos poderosos mecanismos de manipulação de código, normalmente disponíveis neste tipo de ambientes integrados, por forma a facilitar a referência de recursos utilizados nos documentos criados, recorrendo por exemplo aos sistemas de *auto-complete* de código, assim como a consolidar a sua consistência com o projecto de desenvolvimento, por recurso aos sistemas de *refactoring*.

A possibilidade de partilha de acesso a projectos de documentação na aplicação CrystalBox BaseStation entre utilizadores faria sentido, uma vez que a própria ferramenta fomenta a partilha e colaboração na escrita de documentação de software.

A melhoria da interface de utilização ao nível do design e usabilidade seria um último aspecto que poderia ser trabalhado na aplicação CrystalBox BaseStation por forma a potenciar ainda mais a sua facilidade de utilização e consequente adopção.

Por fim, o incentivo à extensão da aplicação com novas representações de recursos e acessos a novos sistemas de controlo de versões através da criação de uma comunidade de utilizadores será crucial para que esta ferramenta cresça e faça sentido na comunidade de desenvolvimento de software.

Bibliography

- 37signals (2009a). 37signals. Disponível em <http://37signals.com/>, acedido a última vez em 21 de Setembro de 2009.
- 37signals (2009b). Basecamp. Disponível em <http://basecamphq.com/>, acedido a última vez em 21 de Setembro de 2009.
- Agile Alliance (2001). Agile software development manifesto. Disponível em <http://www.agilemanifesto.org/>, acedido a última vez em 12 de Setembro de 2009.
- Aguiar, A. (2003). *A minimalist approach to framework documentation*. PhD thesis, Faculdade de Engenharia da Universidade do Porto.
- Aguiar, A. and David, G. (2005). Wikiwiki weaving heterogeneous software artifacts. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*, pages 67–74, New York, NY, USA. ACM.
- Aguiar, A., David, G., and Badros, G. (2004). JavaML 2.0: enriching the markup language for Java source code. Disponível em <http://www.fe.up.pt/~aaguiar/javaml/>, acedido a última vez em 12 de Setembro de 2009.
- Aguiar, A., David, G., and Padilha, M. (2003). XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation. In Pimentel, E., Brisaboa, N. R., and Gómez, J., editors, *JISBD*, pages 11–24.
- Aguiar, A. and Vestdam, T. (2001). Final Report for the 1st meeting at Aalborg University. Technical report, FEUP.
- Aguiar, A. and Vestdam, T. (2002). Final Report for the 2nd meeting at FEUP. Technical report, FEUP.
- Ambler, S. W. (2003a). Agile documentation. Disponível em <http://www.agilemodeling.com/>, acedido a última vez em 12 de Setembro de 2009.
- Ambler, S. W. (2003b). Agile modeling. Disponível em <http://www.agilemodeling.com/>, acedido a última vez em 12 de Setembro de 2009.
- Awesome, L. (2009). Github. Disponível em <http://github.com/>, acedido a última vez em 7 de Setembro de 2009.
- Beck, K. (2000). *Extreme Programming Explained*. Addison-Wesley.
- Beck, K. and Gamma, E. (2009). Junit homepage. Disponível em <http://www.junit.org>, acedido a última vez em 22 de Setembro de 2009.

- Bodner, R. and Chignell, M. (1999). Dynamic hypertext: querying and linking. *ACM Comput. Surv.*, 31(4es):15.
- Bodner, R., Chignell, M., and Tam, J. (1999). Website authoring using dynamic hypertext. In *Proceedings of Webnet'97, Toronto: Association for the Advancement of Computing in Education*, pages 59–64.
- Borland (2003). Borland's together control center. Disponível em <http://www.borland.com/together/>, acessado a última vez em 20 de Setembro de 2009.
- Bray, T., Paoli, J., and Sperberg-McQueen, C. M. (1998). Extensible markup language (xml) 1.0. w3c recommendation. Disponível em <http://www.w3.org/TR/REC-xml/>, acessado a última vez em 12 de Setembro de 2009.
- Cockburn, A. (2004). *Crystal clear a human-powered methodology for small teams*. Addison-Wesley Professional.
- CollabNet (2009). Subversion. Disponível em <http://subversion.tigris.org/>, acessado a última vez em 7 de Setembro de 2009.
- Crockford, D. (2006). JSONRequest. Disponível em <http://www.json.org/JSONRequest.html>, acessado a última vez em 7 de Setembro de 2009.
- Cronin, G. and Barnett, B. (2002-2006). Very Quick Wiki engine homepage. Disponível em <http://www.vqwiki.org/>, acessado a última vez em 12 de Setembro de 2009.
- Cunningham, W. (1999). Portland pattern repository. Disponível em <http://c2.com/cgi/wiki/>, acessado a última vez em 20 de Setembro de 2009.
- Curtis, B. (2009). Ruby on Rails plugins. Disponível em <http://agilewebdevelopment.com/>, acessado a última vez em 7 de Setembro de 2009.
- Delicious (2009). Delicious. Disponível em <http://delicious.com/>, acessado a última vez em 22 de Setembro de 2009.
- DokuWiki (2009). DokuWiki. Disponível em <http://www.dokuwiki.org/>, acessado a última vez em 12 de Setembro de 2009.
- Dunn, B. (2009). Configurator. Disponível em <http://github.com/brennandunn/configurator/tree/master>, acessado a última vez em 6 de Setembro de 2009.
- Escolinhas (2009). Escolinhas – plataforma colaborativa e social para escolas eb1/2. Disponível em <http://escolinhas.pt/>, acessado a última vez em 07 de Outubro de 2009.
- Ferraiolo, J., Jun, F., and Jackson, D. (2003). Scalable vector graphics (SVG) 1.1 specification, w3c recommendation. Disponível em <http://www.w3.org/TR/SVG11/>, acessado a última vez em 12 de Setembro de 2009.
- FEUP/DEEC, U. d. P. (2006a). DocIt! - Cooperative Support for Cost-Effective Development of Quality Framework Documentation. Disponível em <http://doc-it.fe.up.pt/>, acessado a última vez em 12 de Setembro de 2009.
- FEUP/DEEC, U. d. P. (2006b). XSDoc for MoinMoin. Disponível em <http://doc-it.fe.up.pt/moinmoin/>, acessado a última vez em 12 de Setembro de 2009.

- FEUP/DEEC, U. d. P. (2006c). XSDoc for SnipSnap. Disponível em <http://doc-it.fe.up.pt/wiki/space/Projects/XSDoc+for+SnipSnap>, acedido a última vez em 12 de Setembro de 2009.
- Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.
- Flickr (2009). Flickr. Disponível em <http://www.flickr.com/>, acedido a última vez em 22 de Setembro de 2009.
- Flores, N. and Aguiar, A. (2006). Design Pattern Recovery to Improve Framework Understanding. In *1st International Workshop on Design Patterns Detection for Reverse Engineering (DPD4RE 2006), WCRE'06*.
- Foundation, E. (2009a). Eclipse abstract syntax tree. Disponível em http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html, acedido a última vez em 12 de Setembro de 2009.
- Foundation, E. (2009b). Eclipse, an open and extensible integrated development environment. Disponível em <http://www.eclipse.org/>, acedido a última vez em 12 de Setembro de 2009.
- Foundation, W. (2009c). MediaWiki. Disponível em <http://www.mediawiki.org/>, acedido a última vez em 13 de Setembro de 2009.
- Free Software Foundation, I. (1998). CVS. Disponível em <http://www.nongnu.org/cvs/>, acedido a última vez em 7 de Setembro de 2009.
- Git (2009). Git. Disponível em <http://git-scm.com/>, acedido a última vez em 7 de Setembro de 2009.
- Google (2009a). Google. Disponível em <http://google.com/>, acedido a última vez em 21 de Setembro de 2009.
- Google (2009b). Google Docs. Disponível em <http://docs.google.com/>, acedido a última vez em 21 de Setembro de 2009.
- Graphviz (2009). Graphviz - Graph Visualization Software. Disponível em <http://www.graphviz.org/>, acedido a última vez em 12 de Setembro de 2009.
- Hansson, D. H. (2009). Ruby on Rails. Disponível em <http://rubyonrails.org/>, acedido a última vez em 7 de Setembro de 2009.
- Hermann, J. and Waldmann, T. (2009). MoinMoinWiki. Disponível em <http://moinmo.in/>, acedido a última vez em 12 de Setembro de 2009.
- Highsmith, J. A. (2000). *Adaptive software development: a collaborative approach to managing complex systems*. Dorset House Publishing Co., Inc.
- IBM (2003). Jikes java compiler. Disponível em <http://jikes.sourceforge.net/>, acedido a última vez em 12 de Setembro de 2009.
- IBM AlphaWorks (1999). XML metadata interchange (XMI) toolkit. Disponível em <http://www.alphaworks.ibm.com/tech/xmitoolkit>, acedido a última vez em 12 de Setembro de 2009.

- Inc., Y. (2009a). Yahoo! Disponível em <http://yahoo.com/>, acessido a última vez em 22 de Setembro de 2009.
- Inc., Y. (2009b). YUI – The Yahoo! User Interface Library. Disponível em <http://developer.yahoo.com/yui/>, acessido a última vez em 22 de Setembro de 2009.
- Ippolito, B. (2005). Remote JSON – JSONP. Disponível em <http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/>, acessido a última vez em 7 de Setembro de 2009.
- Java 6 API (2009). Java™ Platform, Standard Edition 6 API Specification. Disponível em <http://java.sun.com/javase/6/docs/api/>, acessido a última vez em 12 de Setembro de 2009.
- JSON (2009). JSON – JavaScript Object Notation. Disponível em <http://www.json.org/>, acessido a última vez em 7 de Setembro de 2009.
- Jugel, M. L. and Schmidt, S. J. (2003). SnipSnap Wiki engine homepage. Disponível em <http://www.snipsnap.org/>, acessido a última vez em 12 de Setembro de 2009.
- Knuth, D. E. (1983). The WEB system of structured documentation. Technical Report STAN-CS-83-980, Department of Computer Science, Stanford University.
- Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.
- Knuth, D. E. and Levy, S. (1994). *The CWEB System of Structured Documentation — version 3.0*. Addison-Wesley.
- Kramer, D. (1999). Api documentation from source code comments: a case study of javadoc. In *Proceedings of the 17th annual international conference on Computer documentation*, pages 147–153. ACM Press.
- Langel, T. (2009). PDoc. Disponível em <http://pdoc.org/>, acessido a última vez em 6 de Setembro de 2009.
- M., P. and T., P. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional.
- Merson, P. and Bachmann, F. (2005). Experience using the web-based tool wiki for architecture documentation. Technical report, CMU-SEI.
- MindTouch Deki (2009). MindTouch Deki. Disponível em <http://wiki.mindtouch.com/>, acessido a última vez em 07 de Outubro de 2009.
- Mozilla Developer Center (2009). Mozilla Developer Center. Disponível em <http://developer.mozilla.org/>, acessido a última vez em 07 de Outubro de 2009.
- Mozilla Foundation (2009). The Mozilla Foundation. Disponível em <http://www.mozilla.org/foundation/>, acessido a última vez em 07 de Outubro de 2009.
- Nelson, T. H. (1965). Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, pages 84–100, New York, NY, USA. ACM.
- Nørmark, K. (2000a). Elucidative programming. *Nordic Journal of Computing*, 7(2):87–105.

- Nørmark, K. (2000b). An elucidative programming environment for Scheme. In *Proceedings of NWPER'2000 - Nordic Workshop on Programming Environment Research*, pages 109–126.
- Nørmark, K., Andersen, M., Christensen, C., Kumar, V., Staun-Pedersen, S., and Sørensen, K. (2000). Elucidative programming in Java. In *Proceedings on the eighteenth annual international conference on Computer documentation (SIGDOC)*, pages 483–495. IEEE Educational Activities Department.
- Palmer, J. D. (2009). Ginger: implementing a new lisp family syntax. In *ACM-SE 47: Proceedings of the 47th Annual Southeast Regional Conference*, pages 1–6, New York, NY, USA. ACM.
- Palmer, J. D. and Hillenbrand, E. (2009). Reimagining literate programming. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 1007–1014, New York, NY, USA. ACM.
- Parr, T. (2009). ANTLR Parser Generator. Disponível em <http://www.antlr.org/>, acessado a última vez em 12 de Setembro de 2009.
- Proietti, V. (2009). Mootools - a compact javascript framework. Disponível em <http://mootools.net/>, acessado a última vez em 22 de Setembro de 2009.
- Prototype (2009). Prototype JavaScript framework. Disponível em <http://prototypejs.org/>, acessado a última vez em 6 de Setembro de 2009.
- Pygments (2009). Pygments – python syntax highlighter. Disponível em <http://pygments.org/>, acessado a última vez em 6 de Setembro de 2009.
- Rails API (2009). Rails framework documentation. Disponível em <http://api.rubyonrails.org/>, acessado a última vez em 7 de Setembro de 2009.
- Ramsey, N. (1994). Literate programming simplified. *IEEE Software*, 11(5):97–105. <http://www.cs.virginia.edu/~nr/noweb>.
- RDoc (2009). RDoc – document generator for ruby source. Disponível em <http://rdoc.sourceforge.net/>, acessado a última vez em 6 de Setembro de 2009.
- Reenskaug, T. M. H. (1979). The original MVC reports. Technical report.
- Resig, J. (2009). jQuery. Disponível em <http://jquery.com/>, acessado a última vez em 7 de Setembro de 2009.
- Rueping, A. (2003). *Agile Documentation : A Pattern Guide to Producing Lightweight Documents for Software Projects*. John Wiley & Sons.
- Schwaber, K. (1995). The scrum development process. In *OOPSLA'95 Workshop on Business Object Design and Implementation*.
- Schwaber, K. and Beedle, M. (2002). *Agile Software Development with SCRUM*. Prentice Hall.
- Shepherd, E. (2008). Documenting the mozilla project: A practical example of wikis in open source documentation. In *WikiSym '08: Proceedings of the 3rd Workshop on Wikis for Software Engineering (Wikis4SE 2008)*. ACM.
- Sievers, K. (2009). Gitweb. Disponível em <http://git.or.cz/gitwiki/Gitweb>, acessado a última vez em 20 de Setembro de 2009.

- Software, E. (2009). Trac. Disponível em <http://trac.edgewall.org/>, acessado a última vez em 7 de Setembro de 2009.
- SourceForge (2009). SourceForge. Disponível em <http://sourceforge.net/>, acessado a última vez em 7 de Setembro de 2009.
- Spinellis, D. (2006). UMLGraph. Disponível em <http://www.umlgraph.org/>, acessado a última vez em 12 de Setembro de 2009.
- S.R., P. and J.M., F. (2002). *A Practical Guide to Feature-Driven Development*. Prentice-Hall Inc, Upper Saddle River.
- Stapleton, J. (1999). Dsdm: Dynamic systems development method. In *TOOLS '99: Proceedings of the Technology of Object-Oriented Languages and Systems*, page 406, Washington, DC, USA. IEEE Computer Society.
- Sun Microsystems (2003). Javadoc Tool Home Page. Disponível em <http://java.sun.com/j2se/javadoc/>, acessado a última vez em 12 de Setembro de 2009.
- Thoeny, P. (2009). TWiki™. Disponível em <http://twiki.org/>, acessado a última vez em 12 de Setembro de 2009.
- Tichy, W. F., Habermann, N., and Prechelt, L. (1993). Summary of the dagstuhl workshop on future directions in software engineering: February 17–21, 1992, schloßdagstuhl. *SIGSOFT Softw. Eng. Notes*, 18(1):35–48.
- Twitter (2009). Twitter. Disponível em <http://twitter.com/>, acessado a última vez em 22 de Setembro de 2009.
- van Heesch, D. (2002). Doxygen — a documentation system for c++, java and other languages. Disponível em <http://www.stack.nl/~dimitri/doxygen/>, acessado a última vez em 12 de Setembro de 2009.
- Vestdam, T. (2003). Elucidative programming in open integrated development environments for java. In *Proceedings of the 2nd International Conference on the Principles and Practice of Programming in Java, PPPJ (to be published)*. ACM Press.
- ViewVC (2009). ViewVC. Disponível em <http://www.viewvc.org/>, acessado a última vez em 7 de Setembro de 2009.
- Vitali, F. and Bieber, M. (1999). Hypermedia on the web: what will it take? *ACM Comput. Surv.*, 31(4es):31.
- W3C (2009a). World Wide Web Consortium (W3C). Disponível em <http://www.w3.org/>, acessado a última vez em 6 de Setembro de 2009.
- W3C (2009b). XMLHttpRequest. Disponível em <http://www.w3.org/TR/XMLHttpRequest/>, acessado a última vez em 7 de Setembro de 2009.
- Wunderling, R. and Zockler, M. (1999-2002). Doc++ — A Documentation System for C/C++ and Java. Disponível em <http://docpp.sourceforge.net/>, acessado a última vez em 12 de Setembro de 2009.

- Xiao, W., Chi, C., and Yang, M. (2007). On-line collaborative software development via wiki. In *WikiSym '07: Proceedings of the 2007 international symposium on Wikis*, pages 177–183, New York, NY, USA. ACM.
- YouTube (2009). Youtube. Disponível em <http://www.youtube.com/>, acedido a última vez em 21 de Setembro de 2009.
- Zelkowitz, M. V. and Wallace, D. R. (1998). Experimental models for validating technology. *IEEE Computer*, 31(5):23–31.

