FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Documentação colaborativa de software através de anotações contextuais

**Nuno Miguel Leite Pereira de Sousa**


Mestrado Integrado em Engenharia Informática e Computação

Orientador: Nuno Honório Rodrigues Flores, Doutor

17 de Março de 2015

# Documentação colaborativa de software através de anotações contextuais

**Nuno Miguel Leite Pereira de Sousa**

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente:
Arguente:
Vogal:

17 de Março de 2015

# Resumo

A documentação de software, apesar de nos dias de hoje ainda ser considerada secundária em relação a outros artefactos resultantes do processo de desenvolvimento, desempenha um papel fundamental no uso eficaz e na compreensão do software. O processo de desenvolvimento de software tem sofrido uma evolução ao longo dos tempos, sendo hoje em dia geralmente uma atividade cada vez mais social, nomeadamente quando se olha para frameworks populares na web. Contudo, os mesmos princípios são aplicáveis a equipas mais pequenas e ágeis. Neste contexto, uma documentação minimalista, onde apenas se cria o conteúdo mínimo necessário e que vai evoluindo de acordo com as necessidades dos utilizadores, tem vindo a ser prática cada vez mais utilizada. A interação e comunicação entre leitores e autores, com o intuito de debater que alterações se devem fazer à documentação, é fulcral para essa evolução.

O principal objetivo desta dissertação é mostrar que essa comunicação pode ser melhorada, nomeadamente com o uso de anotações contextuais na documentação. Para o alcançar, foi desenvolvida uma wiki, como plataforma para a documentação colaborativa de software. A essa wiki foi adicionada a possibilidade de adicionar comentários a uma página e anotações a secções específicas, criando um ambiente propício à discussão entre leitores e autores.

Finalmente, de modo a comprovar que a ferramenta desenvolvida cumpre o seu objetivo de melhorar a comunicação entre os vários intervenientes, esta foi avaliada e validada, empiricamente, por uma pequena equipa de desenvolvimento de software.

# Abstract

Software documentation, despite still being considered secondary in relation to other artifacts that result from the development process, is fundamental in order to use and understand software in an efficient way. The software development process has been evolving over time, and is now an increasingly social activity, namely compared to other popular frameworks on the web. However, the same principles can be applied to small agile development teams. In this context, minimalist documentation, where only the bare minimum content is created, which will evolve according to the specific needs of the users, has been rising in popularity. The interaction and communication between both readers and authors, with the goal of debating what changes must be made to the documentation, is key to this evolution.

The main goal of this work is to demonstrate that this communication can be the subject of improvement, namely with the use of contextual annotations in the documentation. In order to achieve this goal, a wiki has been developed, which will serve as platform for collaborative software documentation. Then, the possibility to add comments to a certain page and annotations to specific portions of that page was added to the wiki, thus creating a friendly environment to discussion between readers and authors.

Finally, in order to verify that the developed tool fulfills its goal of improving communication and interaction between the various interveners, it was validated, empirically, in a small software development team.

# Agradecimentos

A realização desta dissertação só foi possível graças ao contributo de várias pessoas.

Ao Prof. Nuno Flores, o orientador deste trabalho, por toda a disponibilidade e conhecimento transmitido. Sem o seu incansável contributo esta dissertação não seria possível.

Ao Prof. Ademar Aguiar e à equipa do Movercado, pelo apoio que prestaram ao longo de todo o projeto.

Ao meu pai, à minha mãe e à minha irmã, por toda a compreensão e influência positiva que transmitiram ao longo do percurso académico.

À Daniela, um especial obrigado por ter estado sempre ao meu lado ao longo da realização desta dissertação, e pela constante motivação.

Nuno Sousa

# Conteúdo

# Lista de Figuras

LISTA DE FIGURAS

# Lista de Tabelas

# Abreviaturas e Símbolos

| | |
|---|---|
| API | Application Programming Interface |
| WYSIWYG | What You See Is What You Get |

# Capítulo 1

# Introdução

Este documento descreve o trabalho desenvolvido no âmbito da Dissertação do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, que culminou no desenvolvimento da Weaki, uma wiki minimalista orientada à documentação colaborativa de *software*.

Neste capítulo será feito um enquadramento do projeto e analisados os motivos que levaram ao seu desenvolvimento. De seguida, serão definidos os resultados esperados desta dissertação e por fim serão resumidos os conteúdos dos vários capítulos deste documento.

## 1.1 Documentação de software

A matéria-prima do desenvolvimento de *software* é o conhecimento [Cor15], e num projeto de engenharia de *software* este é capturado de diversas formas. Um dos artefactos produzidos pelo processo de desenvolvimento de *software* é a documentação, cujo objetivo principal é facilitar o uso e compreensão do *software* por parte de tanto *developers* como utilizadores.

Nos dias de hoje, o processo pela qual a documentação é desenvolvida tem-se tornado cada vez mais *social*, nomeadamente quando se olha para *frameworks* populares de uso generalizado na Web. Neste contexto, a documentação minimalista evolutiva [Agu03] tem sido prática cada vez mais popular, onde é criado o conteúdo mínimo necessário, que posteriormente evoluirá de acordo com as necessidades concretas dos utilizadores. A interação entre autores e leitores torna-se, assim, chave para a produção eficaz de documentação, e é precisamente neste campo que esta dissertação pretende contribuir.

## 1.2 Obstáculos à documentação de software

Apesar das existentes plataformas de documentação colaborativa trazerem já alguma utilidade e facilidade na criação de documentos por parte de vários autores, estas têm ainda espaço para evoluir e melhorar, nomeadamente no que toca à eficaz troca e captura de conhecimento entre os vários intervenientes [CFFA09]. O *software*, nos dias que correm, está em constante evolução,

facto que se alastra à sua documentação. O processo pelo qual esta é criada tende a ser cada vez mais minimalista e evolutivo, isto é, cria-se o conteúdo mínimo necessário, que posteriormente irá crescendo de acordo com as necessidades dos utilizadores [Agu03]. Assim, a possibilidade de poder discutir e partilhar conhecimento no mesmo ambiente onde é criada a documentação seria uma grande mais-valia para todo o processo de desenvolvimento, tornando-o mais célere e eficaz, e é este ponto que será tratado nesta dissertação.

Por outro lado, pretende-se mitigar a perda do conhecimento, da lógica que esteve na base de certas decisões, perda esse que ocorre ao longo da evolução da documentação. Ao manter esse conhecimento próximo do conteúdo da documentação estamos a contribuir para uma melhor compreensão desta, e para uma mais fácil captura desse conhecimento por parte dos intervenientes.

## 1.3 Objetivos e resultados esperados

Por forma a melhorar o processo de criação de documentação, este trabalho tem como objetivo principal melhorar a interação e comunicação entre os vários intervenientes num processo de documentação de *software*, assim como mitigar a perda de conhecimento, que advém da natureza evolutiva da documentação, capturando-o na forma de anotações. Para este fim, será criada uma ferramenta de documentação colaborativa de *software*, utilizando anotações contextuais com o intuito de promover a discussão e a partilha de conhecimento, de *rationale*[1], entre os vários membros de uma equipa.

Espera-se, no fim deste projeto, obter dados que evidenciem que através da documentação colaborativa de *software* utilizando anotações contextuais é possível melhorar todo esse processo, contribuindo assim para uma maturação do próprio processo de desenvolvimento de *software*.

## 1.4 Estrutura do documento

O restante documento encontra-se dividido nos seguintes capítulos:

- Capítulo 2, onde é feita uma revisão do estado da arte e abordados os conceitos mais importantes para a compreensão deste documento.
- Capítulo 3, neste capítulo é definido o problema que este documento pretende atacar. Para esse fim, é proposta uma solução e definido o processo de validação.
- Capítulo 4, onde é descrita a ferramenta desenvolvida e explicado o processo que levou à sua criação.
- Capítulo 5, aborda a validação deste trabalho, explicando o ambiente em que foi levada a cabo, e as suas principais ilações.
- Capítulo 6, neste capítulo são tiradas as conclusões deste trabalho, e feitas sugestões para trabalho futuro.

[1] *rationale*: conjunto de razões ou base lógica que justifica determinada ação

# Capítulo 2

# Estado da arte

Neste capítulo será feita uma revisão do estado da arte. Serão explicados e analisados os conceitos necessários à compreensão deste documento.

Primeiramente, são abordados os conceitos de documentação de software, e os meios que existem para este fim. De seguida, é feita uma pequena análise ao tópico do conhecimento gerado durante a documentação gerado durante a documentação de *software*.

## 2.1 Documentação de software

A documentação de software, apesar de ser muitas vezes relegada para segundo plano comparando com outros processos do desenvolvimento de software, é fundamental para uma boa compreensão e uso do software.

O principal objetivo da documentação de *software* é a captura e partilha de informação acerca de um sistema nas suas diferentes dimensões, incluindo por vezes as atividades relacionadas com a sua criação e desenvolvimento. Documentação de *software* pode ser definida como qualquer forma de informação acerca desse sistema que poderá auxiliar os seus *developers* e utilizadores a compreendê-lo.

Há diversos fatores a ter em conta ao escolher o tipo de documentação a usar:

- **Vários autores.** Na sua grande maioria, a documentação é um processo colaborativo, onde vários intervenientes participam no seu processo de criação. Estes autores poderão ter papéis diferentes no projeto (e.g. programadores, *project managers*, etc.).

- **Públicos-alvo diferentes.** A documentação tem alvos diferentes. Estes alvos podem ser externos ao processo de desenvolvimento do *software*, tal como podem ser os próprios *software developers*. A documentação deve ter em conta que os utilizadores podem ter níveis de conhecimento diferentes.

- **Notações diferentes.** Enquanto que para alguns tipos de informação, uma representação textual será a mais apropriada para uma boa compreensão, outros tipos de informação podem requerer outro tipo de representação, tais como diagramas, gráficos, ou até exemplos de código.

Na próxima secção, 2.2, irão ser analisados vários meios de documentação de *software*.

## 2.2 Meios de documentação de software

Sendo a documentação de *software* definida como qualquer forma de informação acerca de um sistema, esta necessita de um meio onde pode ser criada, armazenada e evoluida. Ao longo desta secção serão abordados os meios de documentação de *software* já existentes mais relevantes no contexto deste trabalho.

### 2.2.1 Wikis

Uma wiki é um sistema que permite a edição colaborativa de páginas Web de uma forma simples. A primeira wiki foi criada em 1995 por Ward Cunningham [Cun], baseada nos seguintes princípios:

- **Aberta.** Se uma página estiver incompleta ou pouco organizada, qualquer leitor pode editá-la da forma que achar melhor.
- **Incremental.** Páginas podem referenciar outras páginas, incluindo páginas que ainda não foram criadas.
- **Orgânica.** A estrutura e conteúdo da wiki estão abertos a alterações e evolução.
- **Mundano.** Um pequeno número de convenções de texto deverão providenciar a formatação necessária.
- **Universal.** Os mecanismos de edição e organização são os mesmos que os da escrita, de modo a que qualquer escritor seja automaticamente um editor e organizador.
- **Evidente.** O *output* formatado deverá sugerir o *input* necessário para o reproduzir.
- **Unificado.** Não deverá ser necessário nenhum contexto adicional para interpretar corretamente os nomes de páginas.
- **Precisa.** As páginas deverão ter títulos suficientemente precisos para evitar conflitos de nome com outras páginas.
- **Tolerante.** Comportamento interpretável, mesmo que indesejado, é preferido em detrimento de mensagens de erro.
- **Observável.** Atividade dentro da wiki pode ser vista por qualquer leitor.

- **Convergente.** Duplicação pode ser desencorajada ou removida encontrando conteúdo semelhante e referenciando-o.

Devido à facilidade que as wikis trazem à edição colaborativa de *software* e à gestão do conhecimento associado, o seu uso como ferramenta de documentação é bastante popular. Algumas das implementações de wikis mais conhecidas são:

**MediaWiki**

A MediaWiki é uma das implementações mais conhecidas, por ser o *software* utilizado pela Wikipedia, além de mais de 2000 sites [Bar08]. Como qualquer outra implementação, esta tem os seus pros e contras:

- **Partilha informal de conhecimento.** Devido à sua filosofia de "qualquer pessoa pode editar qualquer página", a MediaWiki faz um bom trabalho a construir um repositório de conhecimento, sendo mais leve que outros gestores de conteúdo comerciais.
- **Facilidade de uso.** É fácil e rápida de usar. Nem todas as funcionalidades são de aprendizagem rápida, mas uma vez ultrapassado esse obstáculo o utilizador pode pesquisar, modificar e manter os conteúdos de forma eficaz.
- **Fiabilidade.** A MediaWiki é um *software* bastante estável e fiável. Novas versões são testadas durante meses antes de serem oficializadas, eliminando grande parte dos problemas.
- **Controlo de acessos.** A MediaWiki é um sistema público. Apesar de ser possível haver algum controlo de acesso às diversas páginas, este está longe de ser otimizado, e torna-se penoso quanto maior for a wiki, além de não ser muito seguro.
- **Utilizadores leigos.** A MediaWiki requer que os utilizadores aprendam *wikitext*, a linguagem de *markup* utilizada pelo sistema, que permite a formatação do texto (e.g. negritos, itálicos, etc.). Utilizadores menos especializados podem preferir uma wiki com um editor de texto WYSIWYG[1].

**DokuWiki**

A DokuWiki é uma wiki *open-source*, desenvolvida em PHP, orientada à documentação de projetos em pequenas empresas. Algumas das suas funcionalidades básicas são:

- **Sintaxe simples.** A sintaxe é simples e acessível a qualquer utilizador. Por exemplo, para tornar um pedaço de texto negrito, basta rodeá-lo com "**"(ex.: **negrito** é visto como **negrito**);
- **Histórico da página ilimitado.** Sempre que uma página é editada, as versões anteriores ficam gravadas, para o caso do utilizador necessitar visualizar uma versão antiga;
- **Atividade recente.** Mostra uma lista das páginas que foram alteradas recentemente, assim como algumas informações relevantes, tais como o utilizador que fez a alteração, a data e um pequeno sumário daquilo que foi editado;

[1] What You See Is What You Get

- **Upload de ficheiros media.** Permite o envio de imagens e outros ficheiros media para a wiki, para posteriormente incluir em páginas;
- **Extensível.** Permite a personalização da wiki ao nível de temas visuais, ou mesmo a adição de novas funcionalidades. Estas extensões são fornecidas e suportadas pela comunidade, que tem um tamanho considerável;
- **Não necessita de uma base de dados.** Os conteúdos de uma wiki são guardados em ficheiros de texto, pelo que não é necessário configurar uma base de dados

**Gollum**

O Gollum é uma implementação minimalista e simples de uma wiki, no entanto é esse um dos factos que o torna interessante e relevante para este trabalho.

- **Suportado pela comunidade.** O Gollum é desenvolvido e mantido por uma comunidade ativa do Github, e ganhou relevância neste meio por ser a wiki escolhida por este site, com o objetivo de dar aos projetos que aloja um meio simples e eficaz de documentação.
- **Múltiplos formatos permitidos.** O Gollum suporta um conjunto de linguagens *markup*, incluindo Markdown (a linguagem utilizada por defeito), RDoc, MediaWiki e Textile [Gol15].
- **Armazenamento através de Git.** Uma das características que torna o Gollum interessante é a forma como as páginas são armazenadas. Todas as páginas são guardadas num repositório Git presente na máquina onde corre o *software*, o que facilita o controlo de versões e a visualização de páginas em determinados momentos no tempo, usando as ferramentas disponibilizadas pelo Git.
- **Minimalismo.** Outro factor que torna o Gollum um sério candidato a base deste trabalho é a sua simplicidade e minimalismo. Tem praticamente todas as funções básicas necessárias ao funcionamento de uma wiki, sem a sobrecarregar com funcionalidades que neste contexto seriam irrelevantes, possibilitando um produto final feito à medida das necessidades deste projeto.

Abaixo é apresentada uma tabela (2.1) que compara algumas características das três implementações mencionadas acima.

| | **Linguagem de programação** | **Armazenamento** | **Open Source** | **Última *release* estável** | **Controlo de acessos** |
|---|---|---|---|---|---|
| **MediaWiki** | PHP | Sistema de ficheiros, XML | Sim | 26/11/2014 | Muito limitado |
| **DokuWiki** | PHP | Sistema de ficheiros | Sim | 29/09/2014 | Sim, opcional |
| **Gollum** | Ruby | Git | Sim | 22/01/2015 | Sim, via *gem* |

Tabela 2.1: Tabela de comparação entre MediaWiki, DokuWiki e Gollum.

Foram descritas na secção acima algumas das wikis mais populares e relevantes neste contexto, no entanto existem muitas implementações, cada uma com os seus *pros e contras*, o que impossibilita uma análise mais completa neste documento. Assim, caso o leitor pretenda informar-se

acerca de outras wikis que não foram abordadas neste documento, o autor sugere a consulta do WikiMatrix [Wik], um *site* dedicado à indexação e comparação das várias wikis disponíveis.

### 2.2.2 Literate Programming

Proposto por Donald Knuth em 1984, o objetivo de Literate Programming é tornar os programas de computador compreensíveis por humanos alterando o foco principal para artefactos de documentação, em detrimento de artefactos de código, resultando num estilo de programação mais "literário" [Knu84]. Na prática, Literate Programming consiste em combinar documentação, legível e compreensível por humanos, e código fonte, legível pela máquina num único ficheiro. O resultado é um ficheiro com fragmentos de documentação e código, dispostos seguindo uma linha de raciocínio. Os conteúdos ficam dispostos pela ordem em que foram escritos, aumentando a sua compreensibilidade, pois estes seguem a linha de pensamento do *developer* [Knu83].

A implementação de Literate Programming requer o uso de várias linguagens, nomeadamente uma linguagem para a documentação, uma linguagem de programação e uma linguagem de unificação, que auxiliará a combinação das linguagens mencionadas.

Desde o surgimento deste conceito, foram desenvolvidas variadas ferramentas que aplicam estes princípios. No entanto, nenhuma conseguiu grandes níveis de aceitação por parte do público como alguns autores esperavam [PKB04]. A dependência de linguagens específicas de programação e formatação [VAK92], e a falta de integração metodológica no ciclo de vida de um *software* [CB91], foram algumas das barreiras à adopção de Literate Programming.

Alguns dos exemplos mais relevantes de ferramentas de Literate Programming são o `WEB` [Knu83], `CWEB` [KL94] e o `noweb` [Ram94]. Este tipo de ferramentas geralmente funciona da seguinte maneira: existem duas operações fundamentais, `weave` e `tangle`. A primeira é usada para gerar documentação legível e formatada, enquanto que a última é utilizada para gerar código fonte compilável.

A definição de Literate Programming evoluiu ao longo dos anos, tendo-se distanciado um pouco do objetivo inicial, o conceito de produzir documentação semelhante a uma obra literária, aproximando-se hoje em dia a conceitos de organização, integração, recombinação e manutenção da consistência dos conteúdos. Outras técnicas de documentação surgiram a partir de Literate Programming, tais como Anotações no Código (2.2.3) e Elucidative Programming (2.2.4). Estas técnicas serão analisadas nas próximas secções.

### 2.2.3 Anotações no Código

Esta técnica, inspirada inicialmente em Literate Programming (2.2.2), consiste em escrever anotações textuais no código fonte, sob a forma de comentários, que posteriormente irão gerar documentação. Isto significa que tanto o código como a documentação estão num único documento, válido e compilável evitando assim a necessidade de operações do tipo `tangle`. Uma das grandes diferenças entre esta técnica e a sua antecessora é que agora o foco não está no aspecto literário

da programação, mas sim no código. Isto é, o código fonte está organizado de acordo com a sua própria estrutura, e não pela narrativa da documentação.

Este tipo de anotações são usadas, de maneira geral, na documentação de API's e têm tido sucesso neste propósito, aumentando a percepção da necessidade de documentação para uma boa compreensão e usabilidade do *software*.

Várias ferramentas foram desenvolvidas aplicando esta técnica. Algumas das mais relevantes são o Javadoc [Fri96], uma ferramenta que implementa estas anotações na linguagem de programação Java; e o Doxygen [vH], uma das ferramentas mais populares para a linguagem C++, mas que hoje em dia suporta um vasto número de linguagens, tais como C, Objective-C, C#, PHP, Java e Python.

### 2.2.4 Elucidative Programming

Elucidative Programming, um conceito proposto por [Nor00], apesar de inspirado em Literate Programming, tem algumas diferenças fundamentais em relação ao seu antecessor. Elucidative Programming tem como seu principal objetivo providenciar documentação que possa ser usada de forma eficaz dentro do seu próprio ambiente de desenvolvimento. Pode-se dizer que adopta um ponto de vista mais prático, e foca-se na compreensão do programa e nas suas necessidades de manutenção, e não na criação de documentação "literária". EP permite a adição de comentários misturados no código fonte, sem qualquer tipo de prejuízo para a funcionalidade, e sem a necessidade de operações adicionais de `weave` e `tangle`.

Pode-se dividir a criação de documentação por EP em dois estilos [VN02]: o primeiro, **Linked**, consiste na criação de ligações bi-direccionais entre código e documentação externa relacionada, através de hiperligações. Ao navegar no código, o *developer* pode visitar documentação relacionada com determinado segmento, e vice-versa. Neste estilo de documentação, código fonte e documentação são elementos separados, ligados através de referências.

O segundo, **In-line**, consiste em criar fragmentos de texto ao longo do código fonte, sob a forma de comentário. Este estilo surgiu devido ao facto de haver tipos de documentação que beneficiam mais de uma apresentação sequencial dos conteúdos, ao invés de uma leitura usando as ligações vistas no tópico anterior. De qualquer forma, o uso deste estilo não implica que não hajam outros elementos referenciados através de hiperligações.

## 2.3 O Conhecimento no software

Em [Rob99], o desenvolvimento de *software* é descrito como uma atividade altamente baseada em conhecimento, na qual há uma cristalização progressiva do conhecimento com o objetivo de criar um conjunto de instruções que serão executadas por um computador. Desde pedaços soltos de informação, passando por documentos, conversas, discussões até chegar a código fonte e produto final, o conhecimento é trabalhado e partilhado pelos vários membros de uma equipa. Frequentemente o desafio de um projeto de desenvolvimento de *software* não é a produção de instruções numa determinada linguagem, mas sim chegar ao conhecimento que possibilita a sua escrita.

### 2.3.1 Partilha de conhecimento

A partilha de conhecimento e de informação é parte fundamental do processo de desenvolvimento de *software*. Hoje em dia, os *software developers* geralmente fazem parte de uma equipa e, como tal, necessitam trabalhar o conhecimento em conjunto. A partilha eficaz de conhecimento entre os vários membros de uma equipa tornou-se vital, pois permite reduzir o esforço necessário à aquisição de novos conhecimentos, e ajuda a que a equipa trabalhe tendo objetivos comuns bem definidos.

Existem várias formas de partilhar conhecimento, nomeadamente a captura de conhecimento [Cor15], que consiste em guardar conhecimento num certo meio. Ao capturar conhecimento, o autor está torná-lo acessível a outras pessoas e até a si mesmo. Esta prática é importante pois vários fatores podem afetar a manutenção do conhecimento, tais como a entrada e saída de membros da equipa, ou simplesmente o esquecimento de conhecimento adquirido anteriormente. Não se deve, porém, subestimar o valor que a comunicação verbal pode ter, havendo mesmo quem a considere essencial para a partilha de conhecimento a curto prazo [Bec00].

### 2.3.2 Captura de conhecimento

A captura de conhecimento é uma tarefa complexa, e que apresenta vários desafios. Alguns destes desafios são:

- **Grande parte do conhecimento está na mente dos *experts*.** A captura e partilha deste conhecimento aumenta ainda mais o seu valor, embora deva ser apresentado de forma a que *non-experts* o possam compreender.

- ***Experts* têm grande quantidade de conhecimento.** É por isso importante concentrar no conhecimento que é relevante.

- **Cada *expert* não sabe tudo.** O conhecimento deve ser, portanto, acumulado por parte de vários intervenientes, e deve ser permitida a interação entre eles.

- ***Experts* têm muito conhecimento tácito.** Um *expert* por vezes tem mais conhecimento do que aquele que consegue demonstrar. Além de ser difícil de descrever, conhecimento tácito é também difícil de capturar.

- ***Experts* são, normalmente, pessoas ocupadas e cujo tempo é valioso.** As técnicas de captura devem retirar o mínimo de tempo possível aos *experts*.

- **O conhecimento tem "prazo de validade".** O conhecimento evolui, e os *experts* vão descobrindo novo conhecimento. É preciso, portanto, manter e validar o conhecimento ao longo do tempo.

O desenvolvimento de *software* é uma atividade que gere grandes quantidades de conhecimento [Rob99]. Os engenheiros de *software* transformam-se em *learners* quando surge a necessidade de localizar código fonte potencialmente relevante e perceber como o modificar para resolver a tarefa em mãos.

O desenvolvimento de *software* é, também, uma atividade social mediada por artefactos, que são normalmente código fonte, modelos ou documentos textuais. Embora a partilha de conhecimento através da comunicação na primeira pessoa ser importante, o meio principal de transmissão de conhecimento são os artefactos. Assim, maximizando o conhecimento relevante disponível nos artefactos, e apresentando-o de uma forma compreensível para utilizadores de todos os graus de familiaridade, é possível reduzir o tempo dispendido nesta atividade.

## 2.4 Sumário

Neste capítulo foi feita uma revisão do estado da arte, abordando os conceitos mais relevantes no campo em que se insere este trabalho, a documentação colaborativa de *software*. Foi analisado o estado da arte relativamente à documentação de *software*, os meios onde esta é desenvolvida, e o papel do conhecimento no processo de documentação. Após a leitura deste capítulo, o leitor deverá ter as bases necessárias à compreensão do trabalho desenvolvido, e os motivos que levaram à sua criação.

# Capítulo 3

# Documentação de software através de anotações contextuais

Neste capítulo, iremos analisar o problema que este trabalho se propõe a tratar, tendo como base os conceitos já abordados anteriormente no Cap. 2.

Seguidamente será proposta uma solução para esse problema, sendo esta também analisada e documentada nas próximas secções.

## 3.1 Natureza evolutiva da documentação

O *software* está em constante evolução. A incompletude está na natureza dos sistemas de *software*, pois estes normalmente não conseguem responder a todas as necessidades que se espera que resolvam, e de forma geral apenas dão conta dos requisitos mais importantes num determinado momento. Desde o momento em que é idealizado, passando pelo seu desenvolvimento e até mesmo após ser disponibilizado, o *software* é o produto de uma atividade contínua de adaptação, e não uma criação estática, desenhada com um certo objetivo que permanecerá relevante indefinidamente. Alterações nos requisitos surgem constantemente, sejam elas por razões comerciais, políticas ou até derivadas do uso do sistema em si por parte dos utilizadores. A realidade está em constante mudança, tal como a sua percepção por parte dos *stakeholders*.

Assim, a par do *software* também a sua documentação sofre um processo evolutivo. A interação entre os vários intervenientes no processo de desenvolvimento é fulcral para esta evolução, como já foi visto no Cap. 2.

No entanto, ao longo dessa evolução há conhecimento que se vai perdendo, nomeadamente no que diz respeito ao *rationale*, à lógica por trás de certas decisões e alterações. Por outro lado, por vezes a comunicação assíncrona entre autores é um obstáculo quando é necessária a troca de opiniões e o debate sobre alguns aspetos da documentação. Ao capturar este conhecimento, e mantendo-o próximo do conteúdo da documentação, todo o processo se torna mais claro para todos os seus intervenientes.

O objetivo principal deste trabalho é melhorar e facilitar esta interação entre os vários autores da documentação, por forma a melhorar a sua qualidade e, consequentemente, melhorar o processo de desenvolvimento de um *software*. Mais especificamente, pretende-se criar uma ferramenta que possibilite a criação colaborativa de documentação, dando ênfase à discussão e à comunicação entre os autores.

## 3.2 Proposta de solução

Para solucionar o problema mencionado na secção anterior, propõe-se uma ferramenta de documentação colaborativa, podendo esta ser dividida em duas partes principais:

**Documentação colaborativa.**

Sendo a documentação colaborativa de *software* o cerne deste trabalho, foi necessário escolher, e futuramente adaptar, uma plataforma que permitisse este tipo de documentação. Algumas das principais características a ter em conta foram:

- **Criação e edição de páginas.** Sendo a documentação o foco deste trabalho, a possibilidade de criar e editar as páginas que conterão documentação é fundamental.
- **Notação e ligação entre páginas.** A navegação dentro da documentação deverá ser amigável para o leitor, pois serão frequentes as situações em que este precisará de navegar desde a página inicial até chegar à informação pretendida. Por outro lado, a criação de documentação de fácil compreensão para o leitor também agiliza o processo de captura de conhecimento, podendo este objetivo ser alcançado por uma boa formatação de texto. Logo, uma notação que permita a referenciação *inline* de outras páginas, bem como uma fácil formatação do texto é importante para a ferramenta a desenvolver.
- **Motor de busca.** Haverá situações em que o leitor pode necessitar aceder diretamente a uma determinada página, ou procurar por um certo termo, ao invés de navegar pelas várias páginas até atingir a informação que pretende. Assim, um motor de busca que permita ao leitor procurar por certas palavras chave em toda a documentação é também uma funcionalidade importante.
- **Controlo de acessos.** As wikis são geralmente de domínio público. No entanto, no âmbito da documentação de software é importante ter a possibilidade de definir os utilizadores que têm acesso à wiki, pois por vezes a documentação contém informação sensível. De igual modo, é importante também poder definir níveis de acesso (e.g. utilizadores que apenas podem ler certas páginas, outros que podem ler e editar, etc.).

**Comunicação entre autores/leitores.**

Atendendo ao facto que a documentação, a par do *software*, está em constante evolução e que esta é fruto de um processo cada vez mais "social", será acrescentada a possibilidade de adicionar comentários às variadas páginas da documentação, assim como anotações a

secções específicas do texto. Pretende-se com esta adição facilitar a comunicação entre os vários autores e tornar este processo mais amigável, melhorando a qualidade da documentação produzida. O facto de podermos ter lado a lado tanto a própria documentação como as discussões e conversas que a moldaram é uma grande vantagem, pois facilita a percepção do raciocínio que esteve na base do texto que foi produzido aos novos leitores.

Existem outras alternativas no que toca a melhorar a interação entre os vários autores. Chats, por exemplo, são uma solução que permite aos vários intervenientes comunicar entre si em tempo real. No entanto, o facto desta solução necessitar que as pessoas estejam online em simultâneo torna esta alternativa pouco viável. Além disso, seria difícil a transmissão de conhecimento derivado de discussões anteriores a utilizadores que estejam a ler uma página pela primeira vez.

## 3.3 Documentação baseada em wikis

Atendendo aos requisitos estipulados na secção 3.2, o tipo de plataforma escolhido para a ferramenta a desenvolver foi uma Wiki. A escolha recaiu sobre este tipo de ferramenta por esta ser uma solução bastante indicada para o tipo de documentação pretendida, isto é, colaborativa. O facto de uma wiki adequar-se aos requisitos estipulados na secção anterior tornou este tipo de ferramenta um óptimo candidato à implementação da solução ao problema em questão neste trabalho.

Outro factor que foi relevante nesta escolha foi a grande quantidade de wikis *open source* disponíveis, tais como a MediaWiki, DokuWiki ou Gollum. Assim, o foco deste trabalho pôde-se centrar na implementação de um módulo de anotações contextuais, tendo uma wiki reputada como uma base sólida. A escolha da wiki que será utilizada será analisada no capítulo seguinte.

## 3.4 Validação

De modo a obter validação científica que comprove o valor que este trabalho traz à produção de documentação e, por conseguinte, ao processo de desenvolvimento de *software*, é necessário definir os protocolos experimentais que serão utilizados.

Para verdadeiramente compreender a forma como os *software developers* constroem e desenvolvem sistemas de *software*, é necessário ir além das ferramentas e metodologias; é importante investigar também todos os processos cognitivos e o meio social que envolve os indivíduos e a equipa.

Este trabalho tem como objetivo a melhoria da comunicação entre os vários membros de uma equipa de desenvolvimento de *software*, logo qualquer validação tem de demonstrar que este trabalho realmente o conseguiu. Na perspetiva do autor, a maneira mais adequada de validar esta tese seria através de estudos empíricos e experiências controladas, de modo a obter a comprovação necessária. Algumas diretrizes sobre como levar a cabo estudos empíricos podem ser encontradas nas obras de Shull *et al.* [SSS08] e Kitchenham *et al.* [KAKB+08], e as tarefas mais comuns de

um processo de engenharia de *software*, assim como os seus *deliverables* podem ser encontrados em [GA+07].

Assim, a validação desta tese far-se-á através de estudos empíricos, onde se compara a experiência de um grupo que utiliza esta ferramenta com um grupo de controlo que utiliza outra ferramenta já existente. Esta comparação deverá ser feita através de métodos qualitativos (e.g. questionários de satisfação, observação) e quantitativos (e.g. tempo que demora a realizar determinadas tarefas, número de passos necessários, etc.).

## 3.5 Sumário

Neste capítulo foi definido o problema que este trabalho pretende tratar, e seguidamente foi proposta uma solução que consiste na criação de uma ferramenta de documentação colaborativa que ajude a melhorar a interação entre autores e a mitigar a perda de conhecimento. A solução proposta baseia-se numa wiki, à qual serão adicionadas as funcionalidades relevantes para o tratamento do problema em questão. No capítulo seguinte será detalhado o processo de criação da ferramenta supramencionada.

# Capítulo 4

# Weaki

O conceito da Weaki surgiu num artigo de Correia *et al.* [CFFA09], no qual é proposta uma *weakly-typed* wiki orientada ao desenvolvimento de *software*, tendo como principal objetivo uma melhor aquisição de conhecimento dentro de equipas de desenvolvimento de *software*. Assim, e porque este trabalho foi de certa forma inspirado pelos conceitos analisados nesse artigo, o nome dado à ferramenta desenvolvida nesta tese foi Weaki.

A Weaki é uma wiki baseada no Gollum à qual se adicionaram funcionalidades que tornam o seu uso na documentação de *software* mais eficaz e amigável. Nesta secção serão detalhadas as adições que foram feitas ao Gollum, levando à criação da Weaki. Foram várias as alterações, desde pequenos *tweaks* a funcionalidades inteiramente novas, das quais se destacam as seguintes:

- **Controlo de acessos.** Funcionalidade que permite definir uma *whitelist* dos utilizadores que podem aceder à wiki.

- **Anotações contextuais.** Permite adicionar comentários à página, sejam eles gerais ou apenas a secções de texto.

- **Transclusion.** Possibilita a referenciação de métodos e funções de projetos no GitHub, de forma a mostrar sempre o código fonte mais atualizado.

Nas próximas secções serão analisadas cada uma destas novas funcionalidades.

## 4.1 Gollum

Nesta secção será ilustrado o Gollum, a wiki escolhida para servir de base à Weaki. Serão descritos os motivos que levaram à sua escolha, em detrimento de outras ferramentas candidatas.

De seguida será analisada a sua arquitetura, assim como as funcionalidades que foi necessário acrescentar ao Gollum para criar a nossa ferramenta, a Weaki.

### 4.1.1 Porquê o Gollum?

A wiki escolhida para servir de base a este trabalho foi o Gollum, uma wiki já analisada no Cap. 2. Além de servir de forma bastante satisfatória os requisitos referidos anteriormente na secção 3.2, diversos fatores contribuíram para a escolha do Gollum como wiki base deste projeto, em detrimento de outras wikis:

- **Familiaridade.** As linguagens de programação e as tecnologias usadas pelo Gollum tiveram um papel importante nesta escolha. O facto de estar escrito em maioritariamente Ruby e JavaScript tornou-o num candidato apelativo.

- **Minimalismo.** O Gollum cumpre os todos os requisitos para uma ferramenta de documentação colaborativa, sem estar inundado de funcionalidades que seriam irrelevantes neste contexto. A sua simplicidade, sem prejuízo da funcionalidade, facilita a implementação de novas funcionalidades e o suporte a estas.

- **Armazenamento em Git.** O Gollum armazena todas as páginas num repositório Git, alojado na máquina onde este está em execução. Ao utilizar este sistema de controlo de versões, é possível manter versões anteriores de cada página, sendo que cada versão contém o nome do autor e uma pequena mensagem acerca do motivo da edição. Outra vantagem é a possibilidade de editar páginas alterando diretamente o ficheiro e fazendo *commit* das alterações, sem ter de usar o *browser*.

### 4.1.2 Outras ferramentas

Existem outras ferramentas que também encaixavam de alguma forma no perfil definido para esta ferramenta. Entre as diversas plataformas disponíveis para a documentação colaborativa, as seguintes eram as alternativas mais relevantes.

O Google Docs é uma suite de ferramentas de produtividade online, bastante popular por permitir a edição colaborativa de documentos em tempo real por vários autores diferentes. Além disso, possui nativamente um módulo de comentários, que permite criar anotações ao longo do texto, um dos focos principais desta tese. No entanto, apresenta duas grandes desvantagens: *a*) não é *open source*, logo não permite a adição de novas funcionalidades; *b*) é mais indicada para a edição de documentos isolados, pois é algo limitada no que toca a ligações entre documentos e navegação.

A DokuWiki, wiki que já foi revista no Cap. 2, era uma ferramenta que também cumpria os requisitos estipulados. No entanto, foi preterida em relação ao Gollum devido à linguagem de programação em que foi desenvolvida. Tendo em conta o contexto da validação, que seria feita com uma pequena equipa de desenvolvimento de *software* que trabalha maioritariamente com Ruby, decidiu-se optar pelo Gollum, para facilitar a integração, sem prejuízo de funcionalidades.

### 4.1.3 Arquitetura do Gollum

O Gollum, relativamente à sua arquitetura, pode ser dividido em dois grandes módulos: o módulo de *front-end* e o módulo de *back-end*.

O *front-end* é gerido pela *gem* `gollum`[1], à qual é necessário fornecer vários argumentos, como por exemplo: *a*) a porta onde o servidor web irá operar (normalmente a porta 80); *b*) o caminho para o diretório que contém o repositório Git onde estará alojada a wiki. Estes são os argumentos mais relevantes neste contexto, no entanto muitos outros podem ser utilizados consoante a necessidade (e.g. permitir o upload de media, permitir o uso de `mathjax`[2], etc.).

A figura 4.1 mostra um exemplo de uma wiki gerada pelo Gollum.

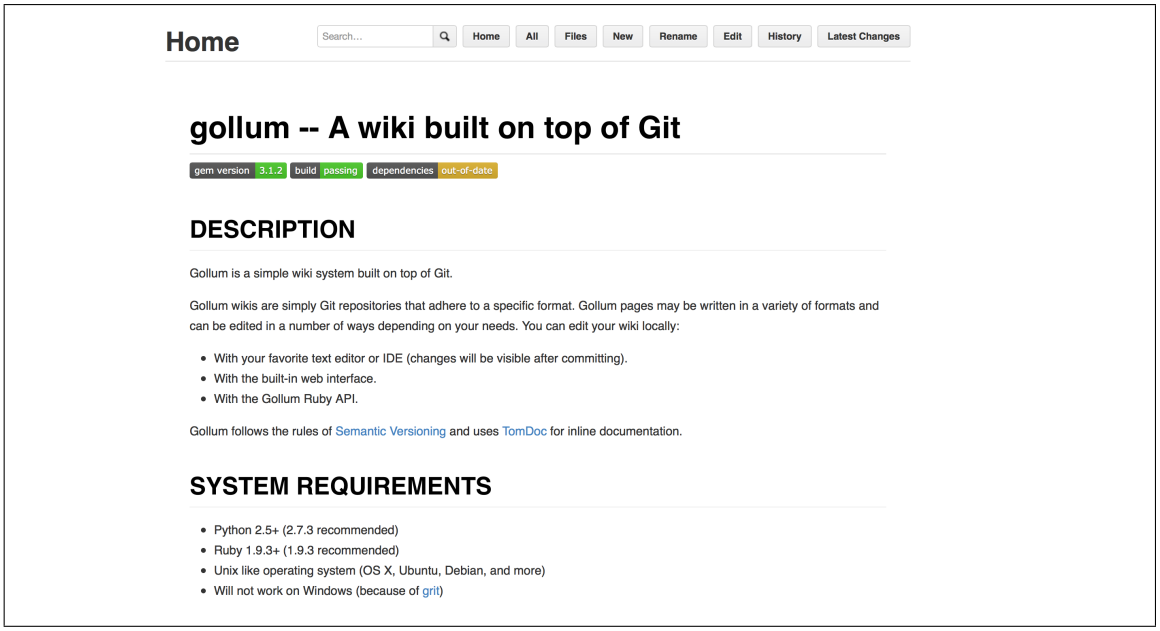


Figura 4.1: Exemplo de uma wiki gerada pelo Gollum.

Por outro lado, o *back-end* é gerido pela *gem* `gollum-lib`[3]. É esta a responsável pela ligação com o repositório Git onde estão alojados os conteúdos da wiki. Todas as operações de leitura e edição de páginas, ou até de visualização das alterações mais recentes à wiki passam por invocações de métodos deste módulo.

Esta separação permite o desenvolvimento de outros *front-ends*, tendo como base o motor de wiki do Gollum. Porém, este não será o caso neste trabalho, visto que o *front-end* disponibilizado serve perfeitamente como base para a Weaki.

O Gollum não necessita nenhuma base de dados, visto que os conteúdos são alojados como ficheiros e pastas num repositório Git presente na máquina onde este está a ser executado. Qualquer configuração ou personalização é feita através de argumentos passados aquando da sua inicialização.

[1] https://github.com/gollum/gollum
[2] MathJax, uma framework JavaScript para a visualização de fórmulas e equações matemáticas no browser
[3] https://github.com/gollum/gollum-lib

No que toca a controlo de acessos, o Gollum por defeito não traz nenhum módulo que forneça esta funcionalidade, sendo uma wiki aberta a quem puder aceder ao seu URL. No entanto, e como será visto na próxima secção, já existe uma *gem* que lhe dá esta importante funcionalidade, permitindo a criação de uma *whitelist*, isto é, uma lista dos utilizadores que podem aceder à wiki.

## 4.2 Controlo de acessos

Tal como foi referido na secção 4.1.3, era importante adicionar a possibilidade de controlar quem pode aceder à wiki. Esta funcionalidade não está presente no Gollum por defeito, mas pôde ser implementada utilizando uma *gem* já existente, o `omnigollum`[4].

Esta *gem* implementa a biblioteca OmniAuth[5] no Gollum, que permite a autenticação com os mais variados *providers*, tais como Facebook, Twitter ou GitHub, utilizando o protocolo OAuth 2.0. Assim, basta o utilizador ter uma conta existente num dos *providers* escolhidos para se autenticar na wiki. No caso da Weaki, foi escolhido apenas o GitHub como meio de autenticação, pois é um *provider* onde um grande número de developers já tem conta e também porque a sua integração fica facilitada no caso da equipa desejar fazer *push* dos conteúdos para um repositório no GitHub.

O modo de funcionamento deste módulo é simples, mas eficaz:

1. Inicialmente é definida uma lista de utilizadores autorizados a aceder à wiki, e o conjunto de *providers* com os quais se podem autenticar, devidamente configurados com as chaves da API do respetivo site.

2. Ao aceder à wiki, o utilizador é forçado a autenticar-se. Caso a autenticação seja bem sucedida e este conste da *whitelist*, é-lhe dado acesso à wiki. Caso contrário, surge uma mensagem de erro.

Apesar de ser uma boa base para um módulo de controlo de acessos, este apresenta alguns problemas no contexto deste trabalho:

- A lista de utilizadores tem de ser definida previamente, e está *hardcoded* no código da *gem*.

- Não é possível especificar a que página(s) o utilizador tem acesso. Ou pode aceder a toda a wiki, ou não pode sequer entrar.

Para atacar estes problemas, foram implementadas algumas alterações a este módulo.

A primeira dessas alterações foi a criação de uma pequena base de dados, para armazenar a informação necessária para o funcionamento deste módulo, nomeadamente a lista de utilizadores com acesso. Esta base de dados foi criada em SQLite, por ser de fácil configuração e perfeitamente capaz para o tipo de uso que lhe seria dado, já que a base de dados teria um tamanho pequeno e,

[4] Página do Omnigollum: https://github.com/arr2036/omnigollum
[5] Página do OmniAuth: http://intridea.github.io/omniauth/

sendo uma wiki orientada a uma equipa, não deverá ter muitos utilizadores a aceder em simultâneo [Sta08, SQL]. A tabela 4.1 apresenta uma breve descrição dos campos presentes na base de dados, e a figura 4.2 mostra o seu diagrama relacional.

Tabela 4.1: Tabelas da base de dados

| **Nome** | **Descrição** |
|---|---|
| Users | Tabela que contém os utilizadores com acesso à wiki |
| Roles | Tabela que contém os vários conjuntos de permissões |
| UsersRoles | Tabela que faz a ligação entre um utilizador e os roles atribuídos |
| Comments | Tabela que contém os comentários da wiki |
| CommentsVersions | Tabela auxiliar que armazena os comentários que estavam presentes numa determinada versão de uma página |

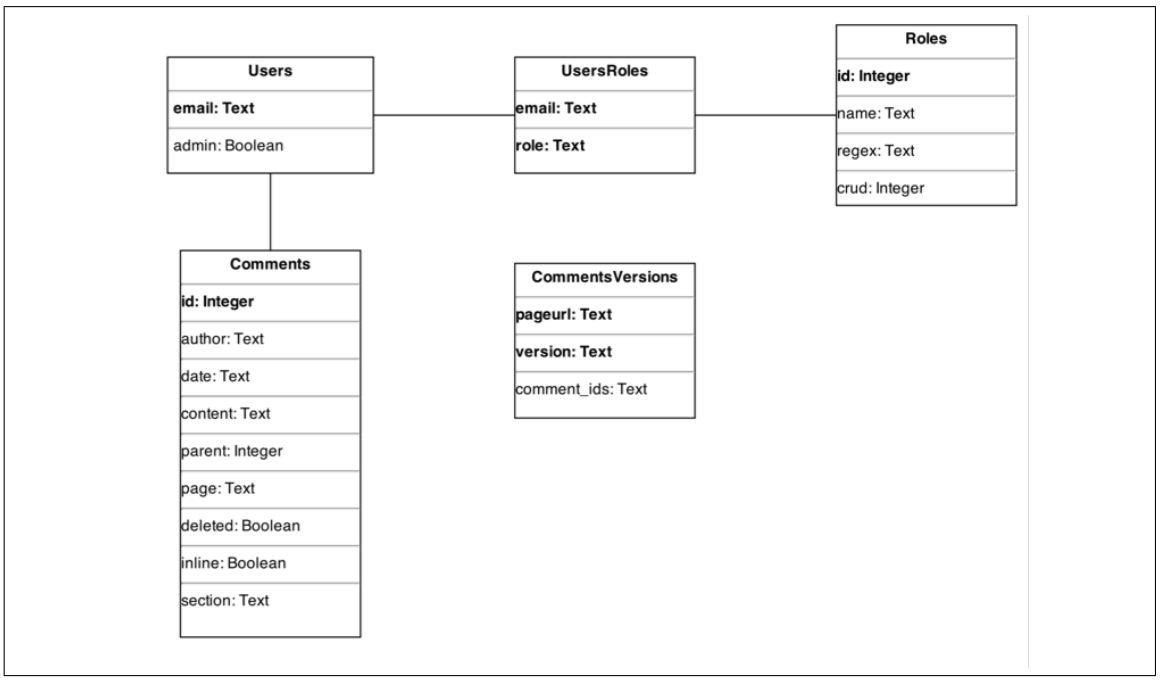


Figura 4.2: Diagrama UML da base de dados auxiliar

Por fim, foi também alterada a forma como é dado o acesso a um utilizador. Introduziu-se o conceito de *role*, isto é, o papel ou função que o utilizador desempenha na wiki, sendo que cada utilizador pode ter um ou mais *roles*. A cada *role* está associado um conjunto de permissões. Estas permissões são compostas por dois campos: uma expressão regular e um nível de permissão. Sempre que um utilizador tenta aceder a determinada página, o caminho desta é comparado com o conjunto de expressões regulares dos seus *roles*, e caso pelo menos uma seja positiva é-lhe concedido acesso a essa página. Por fim, há 4 níveis de permissão sendo eles por ordem crescente: 1) ler, 2) editar, 3) criar e 4) apagar. Cada nível de permissão é cumulativo com os níveis inferiores, ou seja, se foi dada permissão para criar páginas numa determinada pasta, automaticamente também tem permissão para ler e editar nesse mesmo caminho.

Como a lista de utilizadores passou a ser dinâmica, em vez de estar *hardcoded*, foi criada uma pequena página de administração que permite aos administradores da wiki adicionar utilizadores, criar *roles* e atribuí-los.

Com estas alterações pretendia-se melhorar o controlo sobre quem pode aceder à wiki, e ainda a que partes da wiki tem acesso.

## 4.3 Anotações contextuais

As anotações contextuais são o cerne deste trabalho. Estas têm como objetivo criar um ambiente propício à discussão entre os vários autores de uma página da documentação, criando as condições para uma evolução mais rápida e eficaz desta. Nesta secção será detalhado este módulo, a sua estrutura e as suas principais características.

Por uma questão de simplicidade, daqui em diante as anotações contextuais serão referidas como **comentários**.

### 4.3.1 Tipos de comentário

O utilizador tem ao seu dispor dois tipos de comentário que pode adicionar a uma determinada página:

- **Comentário geral à página.** Este tipo de comentário serve para qualquer tipo de anotação que concerne à generalidade da página. Estes comentários são dispostos no fundo da página, após o conteúdo desta.
- **Comentário a uma secção de texto.** Por vezes, o utilizador pretende comentar uma secção de texto específica, seja por esta conter um erro, por não concordar com ela ou por qualquer outro motivo. Assim, este tipo de comentários é disposto do lado direito do texto, estando ao nível da secção a que diz respeito.

Inicialmente, era pretendido que os comentários do segundo tipo estivessem relacionados com os vários parágrafos do texto. No entanto, por questões técnicas, não foi possível alcançar este objetivo. Assim, optou-se por criar o conceito de secção. Uma secção não é mais que um grande pedaço de texto, envolto numa *tag* (`[section] ... texto ... [/section]`) aquando da sua criação. Esta escolha tem a desvantagem de necessitar que o autor delimite as várias secções do seu texto, mas no cômputo geral tem um saldo positivo por permitir a discussão relacionada com secções específicas do texto.

### 4.3.2 Estrutura de um comentário

Os comentários podem ser divididos em dois níveis distintos: comentários à página/secção, ou resposta a um comentário existente. Optou-se por não permitir adicionar respostas a um comentário que já era uma resposta, de modo a não criar uma infinidade de níveis de discussão e a manter a organização e legibilidade do documento.

As informações contidas num comentário são descritas na tabela 4.2.

Tabela 4.2: Campos de um comentário

| **Campo** | **Descrição** |
|---|---|
| Author | Nome do utilizador que criou o comentário |
| Date | Data e hora em que o comentário foi criada |
| Content | Conteúdo textual do comentário |
| Parent | Caso o comentário seja uma resposta, este campo é o identificador do comentário à qual responde |
| Page | A página da wiki onde foi inserido este comentário |
| Deleted | Flag boolean ativada quando este comentário é apagado |
| Inline | Flag boolean que, caso o valor seja *true*, indica um comentário a uma secção |
| Section | Caso seja um comentário a uma secção, este campo indica a secção em questão |

No que toca a armazenamento, os comentários são todas guardados na base de dados auxiliar mencionada na secção 4.2.

### 4.3.3 Casos de uso

No que toca a casos de uso, há várias operações possíveis relacionadas com este módulo. São elas:

- **Inserir comentário à página.** Permite adicionar um comentário à página, num contexto geral.

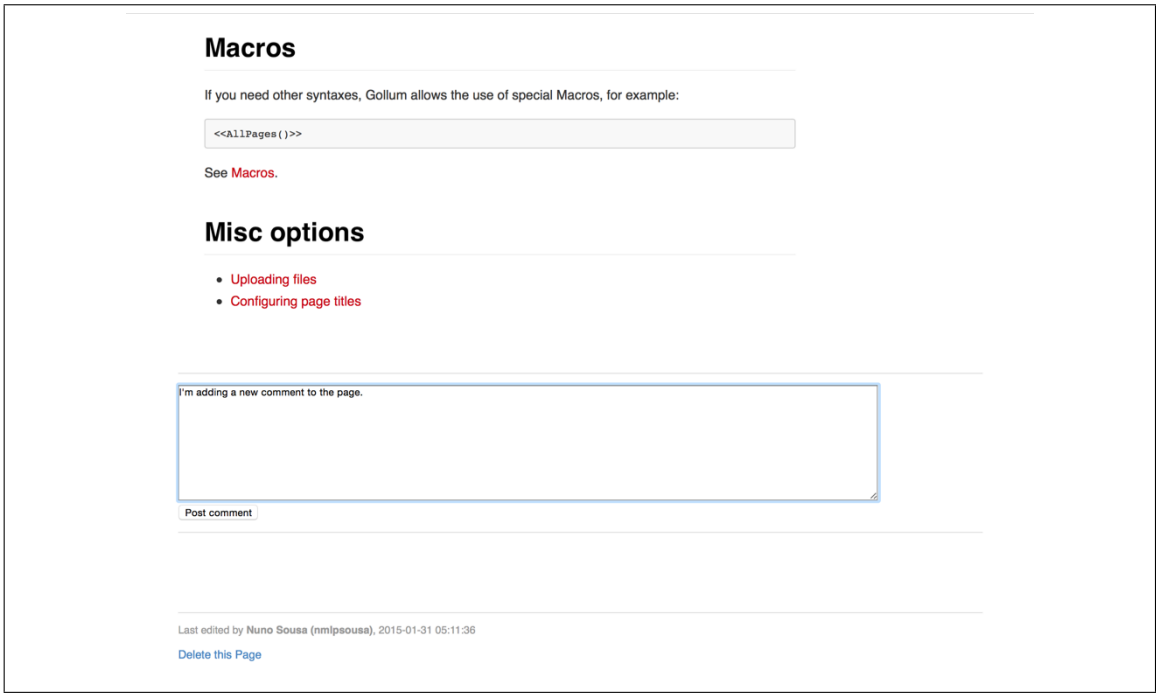


Figura 4.3: Adicionando um comentário à página

- **Inserir comentário a secção.** Permite adicionar um comentário a uma secção do texto, sendo que este aparece ao lado do texto a que se refere.

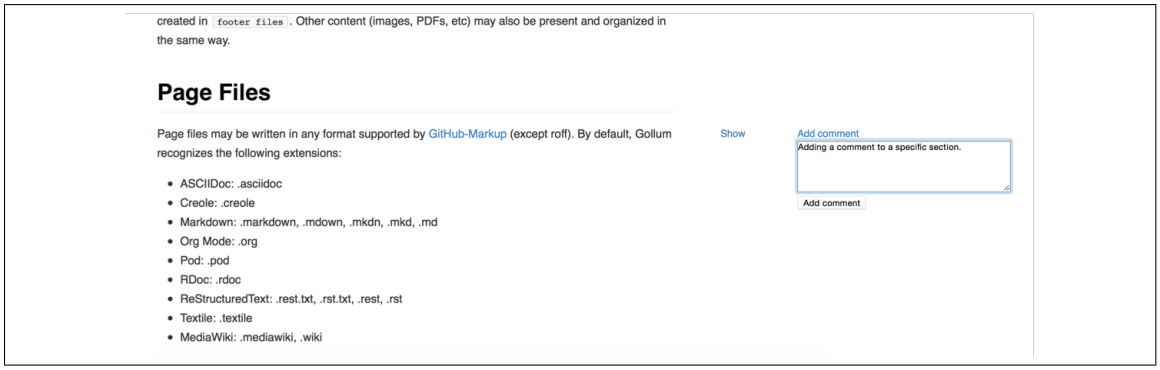


Figura 4.4: Adicionando um comentário a uma secção

- **Responder a comentário da página.** Permite adicionar uma resposta a um comentário já existente.

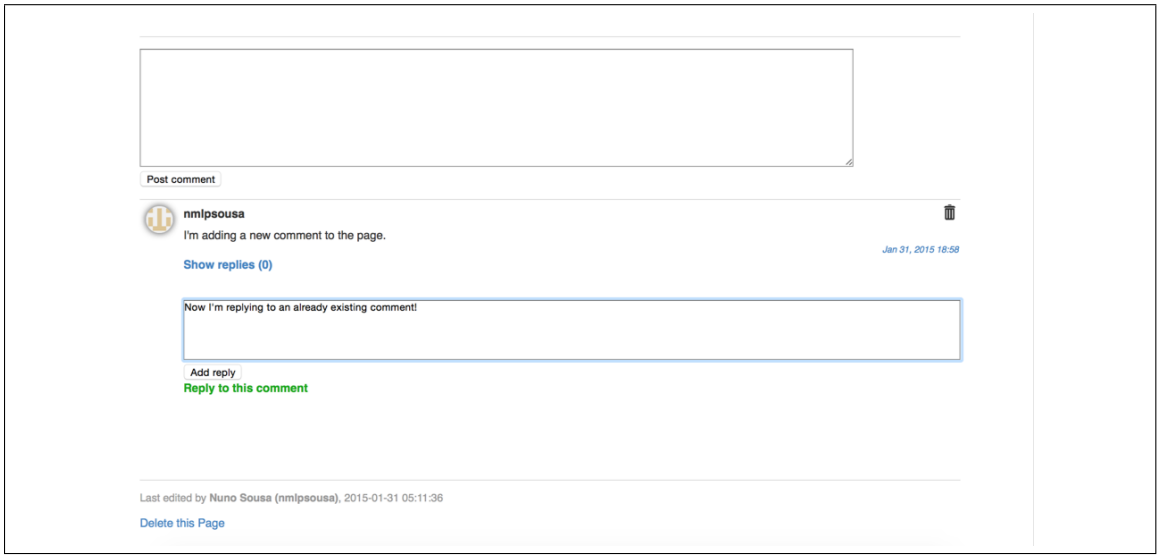


Figura 4.5: Respondendo a um comentário

- **Responder a comentário a uma secção.** Permite adicionar uma resposta a um comentário já existente de uma secção.

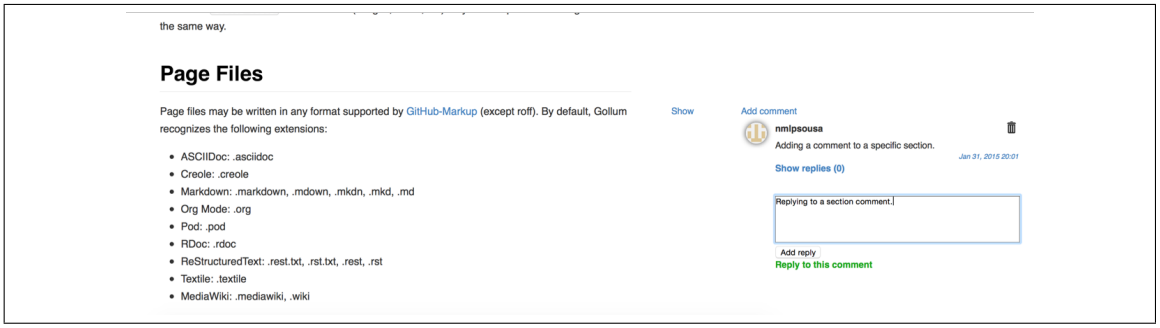


Figura 4.6: Respondendo um comentário a uma secção

- **Visualizar os comentários a uma página.** O utilizador pode visualizar os comentários gerais a uma página, presentes no fim do documento.

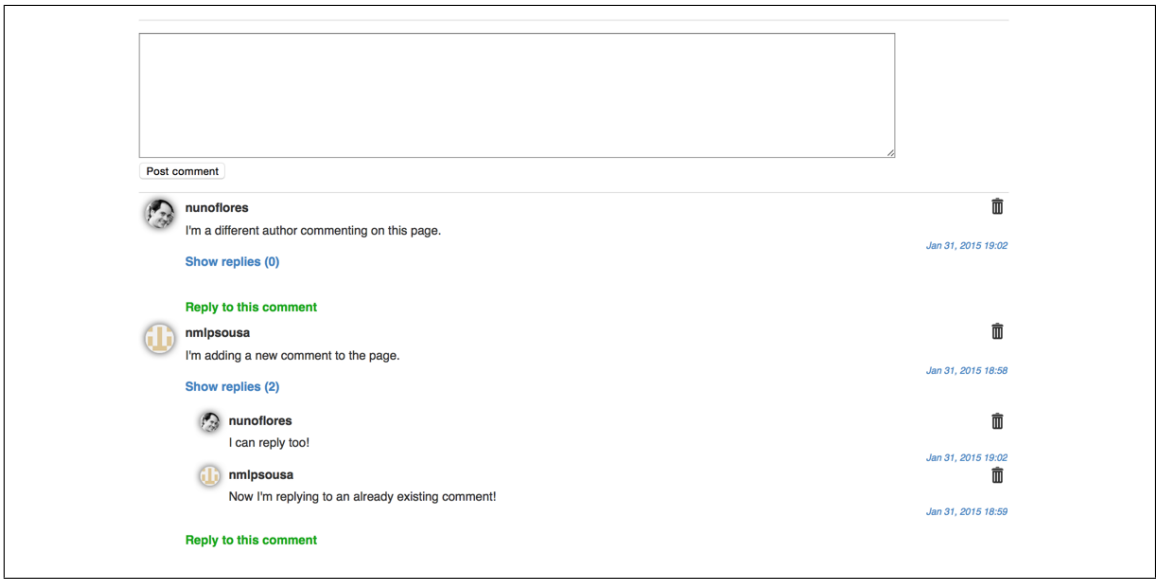


Figura 4.7: Visualizando os comentários gerais de uma página

- **Visualizar os comentários a uma secção.** Clicando no link *Show* visível na figura 4.8, são apresentados os comentários a essa secção.

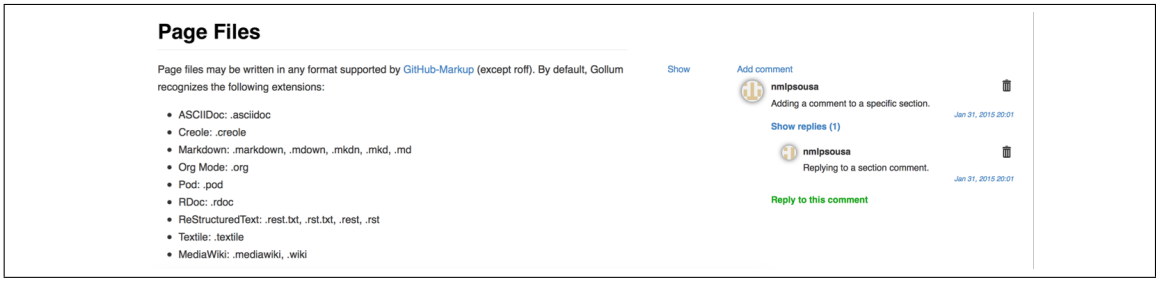


Figura 4.8: Visualizando os comentários a uma secção de uma página

- **Apagar comentário.** Permite apagar um comentário. Esta operação está disponível para o autor do comentário ou questão e para os administradores da wiki, bastando pressionar o ícone semelhante a um caixote de lixo, visível na figura 4.7.

### 4.3.4 Modo de funcionamento (???)

O módulo de comentários pode ser visto como uma camada, uma *layer*, colocada acima da wiki base, isto é, os comentários não interferem com os conteúdos das páginas. Ao abrir uma página, é carregado o seu conteúdo, que está armazenado num repositório *git*, e paralelamente são carregados os comentários relativos a essa página, estando estes armazenados na base de dados auxiliar referida anteriormente.

As várias operações descritas acima geralmente seguem os seguintes passos:

1. O utilizador preenche o campo de texto com o comentário que pretende adicionar. Este campo está inserido num formulário HTML;
2. O formulário é enviado e interpretado pelo servidor da Weaki. O servidor por sua vez executa as operações na base de dados auxiliar;
3. Por fim é recarregada a página.

### 4.3.5 Histórico de comentários

O Gollum permite, nativamente, visualizar cada versão de uma página, dando uso à sua integração com o sistema de controlo de versões Git. Em cada página há um botão *History* que lista as várias versões de uma determinada página da wiki (como se pode ver na fig. 4.9), e clicando na *hash* respetiva é possível visualizar a página tal como era naquele momento.

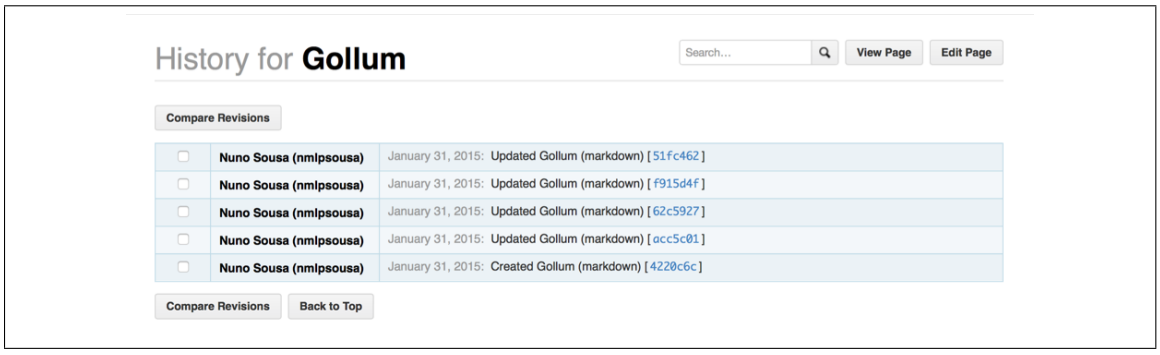


Figura 4.9: Lista de versões da página Gollum

Optou-se, assim, por criar um mecanismo que se integrasse nesta visualização de versões passadas, mostrando os comentários que existiam naquele momento numa determinada página. Tal foi possível armazenando na tabela *CommentsVersions* da nossa base de dados os *IDs* dos comentários que estavam presentes na página, e a *hash* da versão correspondente.

Pretende-se com a adição desta funcionalidade permitir que os utilizadores possam ver não só como era uma página numa versão anterior, mas também a discussão e a interação que estavam presentes nesse momento no tempo.

## 4.4 Transclusão

Transclusão, um termo cunhado por T.H. Nelson [Nel82], refere-se à prática de apresentar conteúdo externo por referência, ao invés de simplesmente duplicá-lo. Este conceito ganha relevância no contexto deste trabalho, pois quando aplicado ao desenvolvimento de *software* permite, por exemplo, referenciar código fonte que está alojado fora da *wiki* (no GitHub por exemplo) e mostrar na página sempre a versão mais atualizada automaticamente, sem a necessidade de intervenção do autor da página sempre que este código fonte sofre alterações.

Assim, foi implementada uma funcionalidade de transclusão na Weaki, que permite referenciar funções escritas em Ruby presentes em repositórios do GitHub. Ao escrever numa determinada página, o autor pode mostrar o código fonte referente a uma função, ao usar uma *tag* específica criada para este efeito.

O modo de funcionamento deste módulo é o seguinte:

1. O autor introduz no conteúdo da página uma *tag* criada especificamente para este módulo (esta *tag* é descrita à frente).

2. Cada vez que um leitor abre a página, uma função Javascript liga-se à API do Github e carrega a versão mais recente da função em questão.

3. Essa função é depois introduzida em linha com o texto, podendo ser omitida ou não consoante o desejo do leitor.

Esta *tag* tem a seguinte estrutura:

```
[method]autor;repo;branch@path#function[/method]
```

Os campos presentes na *tag* são descritos na tabela 4.3.

Tabela 4.3: Campos da *tag* de Transclusão

| **Campo** | **Descrição** | **Exemplo** |
|---|---|---|
| Autor | Nome do autor do repositório | gollum |
| Repo | Nome do repositório | gollum |
| Branch | Branch do projeto onde está a função pretendida | master |
| Path | Caminho para o ficheiro onde está a função | lib/gollum/app.rb |
| Function | Nome da função, precedida pela classe com um "."se for o caso | String.to_url |

A figura 4.10 mostra um exemplo desta funcionalidade em ação. Clicando no nome da função pode-se mostrar/esconder o código fonte associado a esta.

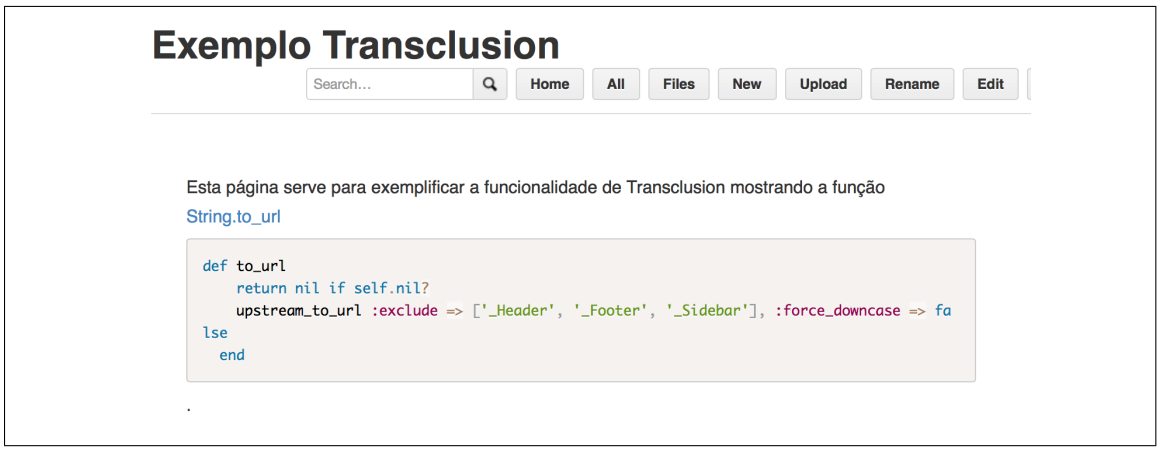


Figura 4.10: Exemplo de Transclusão de uma função.

Com a inclusão desta funcionalidade é facilitada a evolução da documentação, retirando a necessidade de a atualizar sempre que o código fonte de uma determinada função se alterar, algo que é recorrente ao longo de um projeto de *software*.

## 4.5 Metodologia de desenvolvimento

O desenvolvimento da Weaki foi um projeto de engenharia de *software*, tendo sido desenvolvido de acordo com metodologias próprias.

Assim, foi adotada uma metodologia iterativa de desenvolvimento, através de *sprints*. Os *sprints* eram geralmente semanais, sendo que uma vez por semana havia uma reunião com o *scrum master*, que acompanhou regularmente o desenvolvimento deste projeto. Nestas reuniões era revisto o trabalho que tinha sido desenvolvido e definidos os objetivos para o *sprint* seguinte [Sch04]. Para auxiliar este acompanhamento foi utilizado o Pivotal Tracker, uma ferramenta *online* para a gestão de projetos de *software* tendo em conta metodologias ágeis.

Para um melhor controlo de versões, e para poder facilmente ter uma versão estável e outra para testes, todo o trabalho foi desenvolvido sobre um repositório Git, alojado num servidor remoto.

## 4.6 Detalhes de implementação

A Weaki, como já foi visto anteriormente, toma como base a wiki Gollum, sendo que esta foi concebida para correr em ambiente Linux. Assim, a máquina escolhida para acolher a Weaki durante o seu desenvolvimento, para possibilitar testes e até a sua validação foi um servidor Ubuntu 14.04.

A Weaki pode ser vista como três módulos que funcionam em conjunto, sendo que cada módulo corresponde a uma *gem* de Ruby. São eles o **gollum**, o **gollum-lib** e o **omnigollum**.

O primeiro módulo, o **gollum**, é o principal e aquele responsável pelo *front-end* da Weaki e por toda a coordenação com os demais módulos. É neste módulo que se definem todas as páginas que

os utilizadores terão ao seu dispor, e foi também aqui que foi desenvolvido o módulo de anotações contextuais, o foco deste trabalho.

Seguidamente temos o módulo **gollum-lib**, o responsável por toda a interação entre o *front-end* e o repositório Git onde está alojado conteúdo da wiki. Todos os processos de leitura, criação, edição e remoção de páginas passam por este módulo. Este módulo não sofreu grandes alterações em relação ao original do Gollum, visto que as funcionalidades de anotações contextuais e transclusão não necessitavam de mudanças na funcionalidade base do Gollum.

Por último temos o módulo **omnigollum**, módulo responsável pelo controlo de acessos da Weaki. Através deste módulo foi possível criar um mecanismo de controlo de acessos, onde um administrador pode definir que permissões ao nível de leitura/escrita de páginas cada utilizador tem.

### 4.6.1 Operação

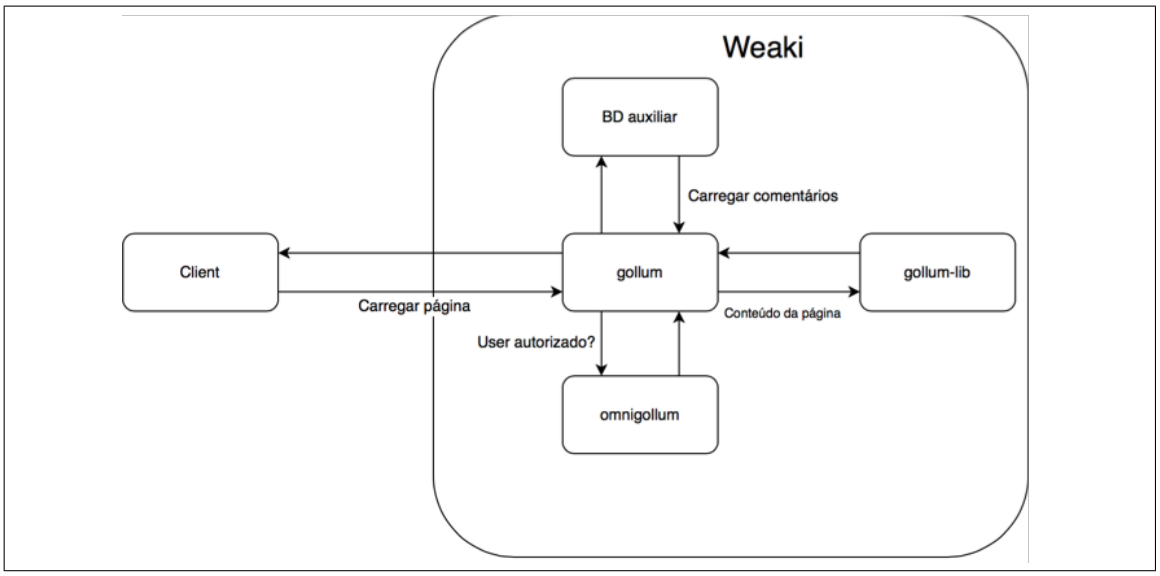


Figura 4.11: Diagrama de fluxo.

Como se pode ver na figura 4.11, será este tipicamente o fluxo de dados no decorrer de uma atividade rotineira, tal como o carregamento de uma página por parte de um utilizador. O **gollum** recebe o pedido do utilizador e começa por verificar, através do **omnigollum**, se este já está autenticado e, caso esteja, verifica também se tem permissões para executar o pedido. Em caso negativo, é apresentada uma mensagem de erro ao utilizador. Caso contrário são utilizadas funções do módulo **gollum-lib** para carregar os conteúdos da página pedida. Posteriormente, é feita uma consulta à base de dados auxiliar para averiguar se a página tem comentários associados, e em caso positivo estes são colocados em redor do corpo da página. Por fim, é enviada a página já com os comentários ao utilizador.

### 4.6.2 Instalação

De modo a criar uma nova instância da Weaki há vários passos que o utilizador tem de seguir. Primeiramente necessita de copiar os repositórios relativos aos módulos necessários para a máquina que a vai executar, sendo esses módulos aqueles vistos na secção anterior. De seguida, o utilizador precisa de inicializar a base de dados auxiliar que servirá de apoio às funcionalidades de controlo de acessos e de anotações contextuais. Esta inicialização pode ser feita através de um pequeno *script* em Ruby criado para esse efeito.

Posteriormente, de modo a possibilitar a autenticação na Weaki através do GitHub, o utilizador precisa de obter uma chave para uso da sua API. Após a sua obtenção, é apenas necessário introduzi-la no ficheiro de configuração *config.ru*, que será utilizado ao executar a Weaki.

É também necessário criar o diretório que irá conter o repositório Git onde estarão os conteúdos da Weaki. Após a criação desta pasta, deve-se correr o comando *git init* com o intuito de criar um repositório Git vazio.

Por fim, o utilizador arranca o servidor dando como argumentos o ficheiro *config.ru* referido acima, que é fornecido, e a porta onde o servidor vai trabalhar (por defeito a porta 80, sendo que esta necessita de privilégios de administrador na máquina).

## 4.7 Sumário

Neste capítulo foi descrita a Weaki, a ferramenta desenvolvida no âmbito deste trabalho. Ao implementar as funcionalidades adicionais ao motor de wiki do Gollum, foi possível criar um ambiente mais propício à discussão e interação entre os vários intervenientes, onde para além da documentação criada é possível capturar toda a lógica que esteve na base da sua criação.

Foram também implementadas outras funcionalidades que contribuem para uma melhor ferramenta no âmbito da documentação de *software*, tais como a Transclusão e o Controlo de acessos.

Contribuiu-se assim, potencialmente, para um melhor processo de documentação, sendo que os dados que evidenciam este facto serão analisados no capítulo seguinte.

# Capítulo 5

# Validação

Este capítulo irá detalhar a forma como foi feita a validação da Weaki. Nas próximas secções será descrito o método escolhido para a validação, revisto o contexto desta validação e detalhada a experiência que foi levada a cabo.

## 5.1 Método de validação

Como já foi referido nas secções anteriores, este trabalho visa melhorar a comunicação e interação entre os intervenientes de um processo de desenvolvimento de *software*. Logo, para poder apurar este facto é necessário recorrer a estudos empíricos. Neste trabalho, optou-se por utilizar principalmente duas metodologias: a observação dos sujeitos enquanto estes trabalhavam com a ferramenta, e questionários para perceber o grau de satisfação relativo a determinados parâmetros. Estes serão detalhados nas secções seguintes.

## 5.2 Contexto

Os participantes desta experiência foram dois engenheiros de *software* envolvidos na documentação de um projeto de uma empresa.

O primeiro, Participante #1, criou a documentação da primeira versão do projeto, sendo aquele que detém um grande conhecimento sobre todo o projeto.

O segundo, Participante #2, ficou encarregado da documentação da segunda versão do projeto, trabalhando com a documentação criado pelo Participante #1 como base. Sendo este Participante novo no projeto, este iria precisar do auxílio do Participante #1 de modo a criar uma boa documentação desta segunda versão.

Ambos os Participantes nunca tinham trabalhado com uma ferramenta deste tipo, isto é, ferramentas de documentação colaborativa de *software*. Este facto é relevante, pois é assim possível aferir que dificuldades têm utilizadores que não tinham experiência anterior.

Temos assim um bom *setup* para executar a nossa experiência, visto que a interação entre os dois intervenientes era fundamental para atingirem os seus objetivos.

## 5.3 Experiência

A experiência foi levada a cabo durante o processo de documentação de um produto real de uma empresa de desenvolvimento de *software*. Como foi referido anteriormente em 5.2, o Participante #1 foi o responsável pela criação da documentação da primeira versão do produto em questão, sendo que na altura ainda não tinha sido desenvolvida a Weaki. Posteriormente, o Participante #2 ficou encarregado de documentar a segunda versão do produto, utilizando como base os conteúdos já anteriormente desenvolvidos. Esta nova documentação já foi criada na Weaki, e pressupunha a necessidade de interação e trabalho em equipa por parte de ambos os Participantes.

Assim, durante alguns dias todo este processo de troca de impressões e comunicação foi observado. Posteriormente, foi elaborado um pequeno questionário para avaliar determinadas métricas do foro qualitativo, e para poder perceber se a Weaki realmente satisfez as necessidades dos utilizadores. Por outro lado, serão também tidas em conta métricas mais quantitativas, tais como o número de comentários registados.

O questionário que foi colocado aos participantes após o período a utilizar a ferramenta pode ser consultado na tabela 5.1.

| **Número** | **Questão** |
|---|---|
| I | Enquanto a utilizava a Weaki, os comentários intrometiam-se demasiado no processo de leitura/escrita de documentação |
| II | Ler, adicionar e responder a comentários foi um processo fácil e intuitivo |
| III | Os comentários ajudaram na interação com outros autores de uma página |
| IV | Foi possível adicionar comentários num contexto suficientemente específico |
| V | Os comentários eram apresentados de uma forma funcional e esteticamente agradável |
| VI | Os comentários foram úteis em todas as situações em que era necessária interação entre autores |
| VII | De uma forma geral, os comentários foram uma experiência agradável |

Tabela 5.1: Perguntas que compunham o questionário

As respostas ao questionário foram dadas utilizando uma escala de Likert. Cada pergunta é respondida com número de 1 a 5, onde 1 corresponde a "Discordo totalmente"e 5 corresponde a "Concordo totalmente". Estas podem ser consultadas na tabela 5.2.

| **Questão** | **Participante #1** | **Participante #2** |
|---|---|---|
| I | 2 | 1 |
| II | 5 | 5 |
| III | 5 | 4 |
| IV | 5 | 5 |
| V | 4 | 5 |
| VI | 4 | 4 |
| VII | 4 | 5 |

Tabela 5.2: Respostas ao questionário

## 5.4 Análise dos resultados

Como se pôde verificar na secção anterior, os dados recolhidos foram promissores. As respostas positivas ao questionário evidenciam que as anotações contextuais no âmbito da documentação colaborativa de *software* têm potencial para agilizar e tornar mais eficaz o processo de desenvolvimento, ao facilitar a interação entre os vários intervenientes.

Ao observar os dois participantes a utilizar a ferramenta, foi perceptível a sua facilidade de uso mesmo para quem não tinha trabalhado com ferramentas daquele tipo, facto que é corroborado pelas respostas ao questionário. Convém salientar que as respostas à questão *I* fugiram da norma, no sentido em que quanto mais baixa a resposta, melhor.

Contudo, foi também possível observar desde já algumas potenciais melhorias a implementar de futuro na ferramenta, nomeadamente um sistema que notifique um utilizador caso tenham sido adicionados comentários a uma página da qual é autor.

Concluindo, os dados recolhidos por este pequeno estudo empírico sugerem que a documentação colaborativa de *software* utilizando anotações contextuais contribuem para uma melhor experiência dos seus intervenientes.

## 5.5 Estudos futuros

Apesar das evidências positivas levantadas pela experiência descrita anteriormente, é do entendimento do autor que um estudo mais alargado, contando com mais participantes e uma janela temporal maior são importantes para corroborar os dados recolhidos e obter uma validação mais forte. Assim, estão a decorrer estudos com equipas maiores para este fim. Infelizmente, não foi possível obter resultados desses estudos atempadamente.

## 5.6 Sumário

Neste capítulo foi detalhada a Validação, o método pela qual esta foi obtida, e apresentados os detalhes do estudo levado a cabo, assim como uma análise dos seus resultados.

Os dados recolhidos evidenciam uma contribuição positiva da ferramenta para o processo de documentação, apontando também para uma satisfação dos objetivos propostos no início deste trabalho.

No capítulo seguinte serão apresentadas as conclusões retiradas deste trabalho, sendo também descritas algumas diretrizes para trabalho futuro.

# Capítulo 6

# Conclusões

Neste capítulo serão apresentadas as conclusões retiradas deste projeto, assim como algumas direcções de trabalho futuro.

## 6.1 Satisfação dos objetivos

No início deste documento, foram estipulados como objetivos principais desta tese:

1. a criação de uma ferramenta de documentação colaborativa de *software* através de anotações contextuais,

2. levantamento de evidências que mostrem que utilizando anotações contextuais é possível melhorar o processo de desenvolvimento de *software*.

No final deste projeto, pode-se concluir que estas metas foram atingidas. Foi desenvolvida a Weaki, uma wiki orientada ao desenvolvimento de *software* que conta com um arsenal de funcionalidades cujo intuito é tornar mais eficaz e ágil a produção de documentação. Implementando um módulo de comentários numa wiki, foi possível criar uma ferramenta onde a produção de documentação por parte de vários intervenientes tornou-se mais rápida, eficaz e amigável. Através de uma pequena experiência empírica, observou-se a colaboração entre dois intervenientes na documentação de *software* e encontraram-se indícios de que, o uso de anotações contextuais promove uma melhor e mais focada comunicação e partilha de conhecimento. A utilização dos comentários promoveu a partilha de conhecimento e de *rationale* entre os vários utilizadores, culminando num melhor processo de desenvolvimento de *software*.

## 6.2 Trabalho futuro

Apesar deste trabalho ter atingido os objetivos propostos, existem ainda vários pontos de melhoramento e de trabalho futuro que podem aumentar o contributo desta ferramenta no campo da documentação de *software*. De seguida, serão dados alguns exemplos de trabalho futuro:

- **Possibilidade de comentar parágrafos ou pedaços de texto automaticamente.** Uma das limitações que foi discutida no Cap. 4 foi a necessidade de delimitar manualmente as secções de texto. A possibilidade de comentar parágrafos e pedaços de texto sem ser necessário intervir no conteúdo do documento é uma prioridade para o futuro.

- **Notificações de comentários e alterações.** Uma das sugestões recolhida durante o processo de validação foi a possibilidade de receber notificações quando uma página da qual o utilizador é autor é alterada, ou comentada. Esta sugestão pode se expandir para a criação de uma plataforma de conhecimento mais "social", onde um utilizador poderia, através de comentários, requisitar a atenção de outro(s) utilizador(es) mencionando-o(s) no conteúdo do comentário.

- **Tratamento dos comentários como sugestões a aceitar/rejeitar.** Quando um utilizador, seja ele um autor ou apenas um leitor, adiciona um comentário cujo intuito é sugerir uma alteração ao conteúdo da página seria benéfica a possibilidade de aceitar ou rejeitar essa alteração, fornecendo a razão para esta decisão.

- **Testes de aceitação.** A possibilidade de executar, através da wiki, testes de aceitação no código fonte do projeto seria uma mais valia. Assim, os utilizadores podem definir testes que ilustrem determinados requisitos, e obter os resultados automaticamente através da documentação.

Para finalizar, e reforçando o que foi dito na secção 5.5, a realização de outros estudos empíricos com mais e maiores equipas e em ambientes multi-disciplinares seria importante, tendo como objetivo a corroboração dos dados que foram recolhidos na experiência descrita no Cap. 5, por forma a validar o contributo desta ferramenta com um maior grau de confiança.

# Referências


[Agu03] Ademar Aguiar. *A minimalist approach to framework documentation*. PhD thesis, 2003.

[Bar08] Daniel J Barrett. *MediaWiki*. "O'Reilly Media, Inc.", 2008.

[Bec00] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[CB91] David Cordes e Marcus Brown. The literate-programming paradigm. *Computer*, 24(6):52–61, 1991.

[CFFA09] Filipe F Correia, Hugo S Ferreira, Nuno Flores e Ademar Aguiar. Incremental knowledge acquisition in software development using a weakly-typed wiki. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, page 31. ACM, 2009.

[Cor15] Filipe Figueiredo Correia. *Documenting Software With Adaptive Software Artifacts*. PhD thesis, 2015.

[Cun] Ward Cunningham. c2.com - wiki. http://c2.com/cgi/wiki. Accessed: 23/01/2015.

[Fri96] Lisa Friendly. The design of distributed hyperlinked programming documentation. In *Hypermedia Design*, pages 151–173. Springer, 1996.

[GA+07] Miguel Goulão, Brito E Abreu et al. Modeling the experimental software engineering process. In *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, pages 77–90. IEEE, 2007.

[Gol15] Gollum. Gollum wiki page. https://github.com/gollum/gollum/wiki, 2015. Accessed: 14/01/2015.

[KAKB+08] Barbara Kitchenham, Hiyam Al-Khilidar, Muhammed Ali Babar, Mike Berry, Karl Cox, Jacky Keung, Felicia Kurniawati, Mark Staples, He Zhang e Liming Zhu. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, 13(1):97–121, 2008.

[KL94] Donald Ervin Knuth e Silvio Levy. *The CWEB System of Structured Documentation: Version 3.0*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[Knu83] Donald Ervin Knuth. *The WEB system of structured documentation*. Department of Computer Science, Stanford University, 1983.

[Knu84] Donald E Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[Nel82] T Nelson. *Literary Machines*. Eastgate Systems, 1982.

[Nor00] K Normark. Requirements for an elucidative programming environment. In *Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on*, pages 119–128. IEEE, 2000.

[PKB04] Vreda Pieterse, Derrick G Kourie e Andrew Boake. A case for contemporary literate programming. In *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 2–9. South African Institute for Computer Scientists and Information Technologists, 2004.

[Ram94] Norman Ramsey. Literate programming simplified. *IEEE software*, 11(5):97–105, 1994.

[Rob99] Pierre N Robillard. The role of knowledge in software development. *Communications of the ACM*, 42(1):87–92, 1999.

[Sch04] Ken Schwaber. *Agile project management with Scrum*. Microsoft Press, 2004.

[SQL] SQLite. Appropriate Uses For SQLite. http://www.sqlite.org/whentouse.html. Accessed: 23/01/2015.

[SSS08] Forrest Shull, Janice Singer e Dag IK Sjøberg. *Guide to advanced empirical software engineering*, volume 5. Springer, 2008.

[Sta08] Justin Standard. When to use SQLite. http://stackoverflow.com/a/3632/1025725, 2008. Accessed: 23/01/2015.

[VAK92] Eric W Van Ammers e Mark R Kramer. Vamp: A tool for literate programming independent of programming language and formatter. In *CompEuro'92.'Computer Systems and Software Engineering', Proceedings.*, pages 371–376. IEEE, 1992.

[vH] Dimitri van Heesch. Doxygen home page. http://www.stack.nl/~dimitri/doxygen/. Accessed: 12/01/2015.

[VN02] Thomas Vestdam e Kurt Normark. Aspects of internal program documentation-an elucidative perspective. In *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, pages 43–52. IEEE, 2002.

[Wik] WikiMatrix. Wikimatrix. http://www.wikimatrix.org. Accessed: 12/01/2015.