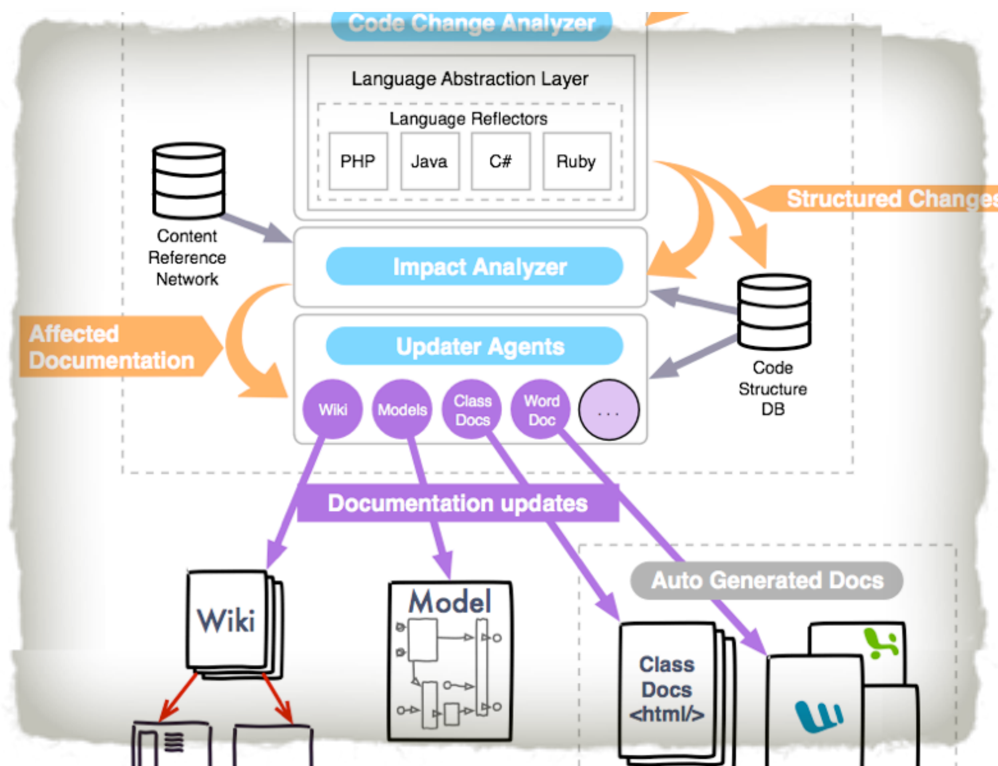
SOFTDOC



A Software Documentation Platform

Saulo Vallory
Summer 2009

TABLE OF CONTENTS

Resumo 3

Palavras-chave 3

Introdução 4

A documentação em frameworks 5

Analisando o problema 7

Arquitetura 11

- *Módulo Code Watcher* 12
- *Módulo Code Change Analyzer* 12
- *Módulo Impact Analyzer* 12
- *Módulo Updater Agents* 13
- *Camada de apresentação* 13
- *Updater Agentes + Camada de apresentação* 15

Discussão 16

Bibliography 17

SOFTDOC

Uma Plataforma para Documentação de Software

Saulo Vallory
Summer 2009

Resumo

O SoftDoc pretende tornar a documentação de *software* uma tarefa mais *leve (soft)*. Criar uma boa documentação de um software requer uma grande quantidade de documentos que devem se dirigir a vários grupos, usando linguagem e formato apropriados. Esta tarefa torna-se ainda mais complicada quando o software começa a evoluir e, em decorrência desta evolução, a documentação, por vezes vasta, precisa ser alterada para refletir as mudanças no software. Este artigo procura levantar os principais problemas envolvidos na criação da documentação e definir as características de uma plataforma capaz de resolvê-los.

Palavras-chave

software, documentation

Introdução

O campo de engenharia de software sempre buscou estabelecer práticas eficientes para o desenvolvimento e manutenção de software. Entre essas práticas, a documentação sempre foi vista como facilitadora do desenvolvimento e necessária à manutenção. Apesar disso, a documentação dos softwares produzidos é geralmente incompleta, obsoleta e as vezes quase inexistente.

Nos últimos anos, uma outra visão de documentação tem ganhado força. Metodologias ágeis de desenvolvimento dão pouca, ou nenhuma importância aos documentos que outros métodos pregam ser tão importantes quanto o software em si. A metodologia XP (*Extreme Programming*) confia exclusivamente na "comunicação oral, testes e código-fonte para comunicar os objetivos e estrutura de um sistema" (Beck 2001).

Há entretanto uma classe de software onde a documentação é reconhecida como indispensável, acredito, até mesmo pelos mais devotos praticantes de metodologias ágeis: frameworks. "O mais profundamente elegante framework nunca será reusado a não ser que o custo de entendê-lo e usar suas abstrações seja menor que o custo percebido pelo programador de escrevê-las do zero" (Booch 1994).

Não é o objetivo desta pesquisa provar a necessidade da documentação de frameworks. Mas, partindo do pressuposto de que tais documentos são desejáveis, estabelecer uma plataforma que auxilie a criação dessa documentação, reduzindo o custo tanto da criação quanto da manutenção dos documentos que a compõem. É importante ressaltar que, apesar da freqüente referência a frameworks, softwares com o qual o autor tem mais contato, a plataforma aqui definida pode ser utilizada para a documentação de qualquer tipo de software.

Com este propósito em mente, pode-se observar, como exposto no capítulo seguinte, que o estudo da documentação de frameworks é particularmente útil. Este tipo de software é significativamente mais difícil de ser documentado devido ao grande número de formas de reuso (Butler, Keller et al. 2000) e aos diferentes níveis da audiência a qual a documentação deve se dirigir (usuários novatos, experientes, desenvolvedores do framework etc.). Todos esses documentos exigirão um grande esforço de manutenção a medida que software evolui. É necessário um esforço ainda maior para identificar e manter as correlações entre eles, ou entre a documentação e o código-fonte.

A documentação em frameworks

Como disse Booch, "O mais profundamente elegante framework nunca será reusado a não ser que o custo de entendê-lo e usar suas abstrações seja menor que o custo percebido pelo programador de escrevê-las do zero" (Booch 1994). Mas porque isso é verdade para frameworks quando freqüentemente utilizamos outros softwares sem o auxílio de qualquer documentação?

Diferentemente dos softwares que utilizamos com mais freqüência, frameworks não possuem interfaces gráficas que nos permitam inspecionar suas "ferramentas" (*hotspots*). A única maneira de conhecer um framework, sem recorrer à sua documentação, seria ler o código fonte. Tarefa impraticável para muitos, talvez grande maioria, dos frameworks existentes, dado o volume e a complexidade de seus códigos-fonte. Em seu artigo "A Framework for Framework Documentation", Butler atribui a necessidade da documentação de um framework ao fato de que, no caso de frameworks, "o *design* é incompleto, requerendo novas subclasses para criar uma aplicação; o *design* provê flexibilidade para alguns *hotspots*, nem todos necessários para a aplicação sendo desenvolvida; e as colaborações e as dependências criadas por elas entre as classes podem ser indiretas e obscuras" (Butler, Keller et al. 2000).

Antes de prosseguirmos, estabeleçamos, para fim de praticidade, que um público-alvo da documentação será definido não pelas pessoas que a utilizarão, nem por seus cargos, mas pelo papel que desempenharem no momento em que estiverem utilizando o software. Ou seja, um programador pode pertencer aos públicos-alvo: avaliador (se está escolhendo um framework), usuário novato (se está dando os primeiros passos com o *framework*) ou usuário avançado.

Alguns softwares (e.g. um editor de textos) são desenvolvidos com um único uso em mente. Ainda assim, a documentação desses softwares pode ter vários públicos-alvo, como, por exemplo, um administrador de sistemas que deseja instalar o software em diversas máquinas e um usuário que quer instalar ou apenas realizar uma tarefa com o editor.

Se atrelamos o público-alvo da documentação às formas de uso do software, precisamos, antes de pensar sobre a documentação de frameworks, entender como eles são usados. Se utilizarmos a categorização proposta por Butler (Butler, Keller et al. 2000) temos seis formas de (re)uso:

Uso comum

Selecionando: Um framework é selecionado como apropriado para a aplicação a ser desenvolvida.

Instanciando: Um framework é instanciado para um problema particular selecionando-se classes concretas para os *hotspots* de uma biblioteca existente, ou estendendo-se classes parcialmente abstratas em uma das formas planejadas pelo criador do framework.

Uso estendido

Flexibilizando: Um *hotspot* é utilizado de uma maneira que não foi planejada pelo criador do framework, mas a utilização é consistente com as obrigações e restrições do *design*.

Combinando: Dois frameworks são usados em conjunto, possivelmente compartilhando integrantes e mensagens ou seqüências de mensagens (Mili, Sahraoui et al. 1997).

Manutenção e reuso

Evoluindo: Um framework é mantido para aumentar a flexibilidade de *hotspots*, ou para adicionar novos *hotspots*.

Minerando: Um framework é minerado para encontrar idéias que podem ser aplicadas em outros contextos, como um framework ou um sistema para outro domínio (Keller, Tessier et al. 1998).

Entre os usos acima, a instanciação do framework deve ter um tratamento especial. Dada a complexidade de muitos frameworks, é interessante separar esta forma de uso em pelo menos dois públicos: usuários novatos e usuários experientes. Essa divisão permite criar uma documentação mais didática que guiará o leitor no aprendizado de forma gradual.

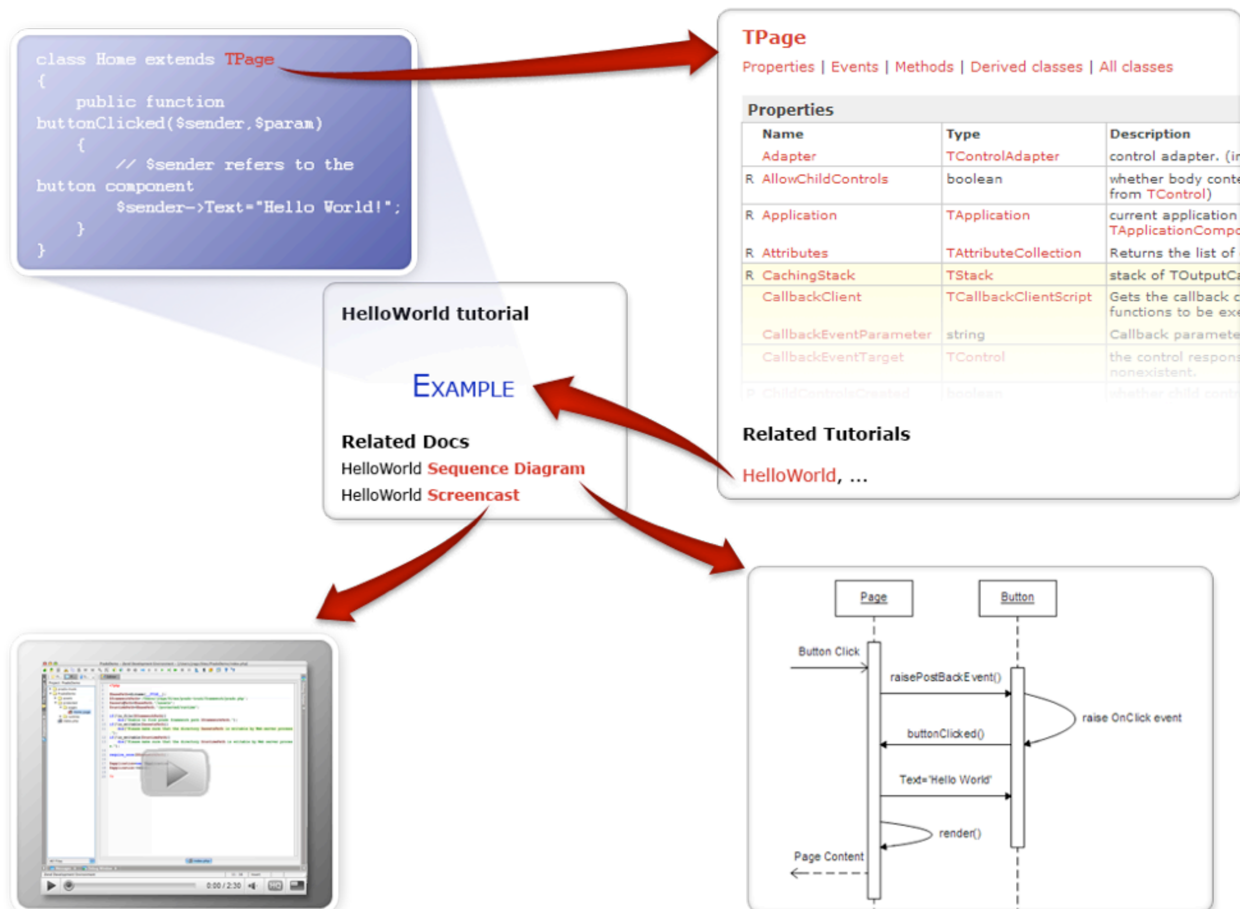
Ainda no brilhante trabalho de Butler, podemos ver que para cada um desses casos há um formato de documentação mais adequado. Conforme ele mesmo apresenta no seguinte quadro:

Documentation approach	Framework (re-)use case					
	Select	Instantiate	Flex	Compose	Evolve	Mine
Example applications	•	•				
Recipe / cookbook	●	●				
Interface contract		•	•	•		
Interaction contract		•	●	●	●	
Design pattern			●		●	●
Framework overview	●					•
Reference manual		•	•		•	

Table 3. Documentation Approaches to Framework (Re-)use (Butler, Keller et al. 2000)

A conclusão de Butler é que "o desafio é construir uma documentação que seja ortogonal, auto-referenciada, e econômica, tanto para construir quando para compreender".

Abaixo podemos ver um exemplo do que "auto-referenciada" significa para a documentação de um framework.



Exemplo de como diferentes artefatos de documentação podem ser relacionados

Temos em mãos um cenário onde a documentação cobre não uma aplicação concreta, mas uma ferramenta para produzir uma família de muitas aplicações concretas semelhantes (Aguiar, David et al.), deve se dirigir a diferentes audiências e evoluir a medida que o próprio software evolui. O capítulo seguinte se aprofunda nesse cenário tentando dar luz aos principais problemas que permeiam a tarefa de documentação de software. A partir daí, poderemos estabelecer quais são os princípios que deverão guiar o desenvolvimento do SoftDoc.

Analisando o problema

A documentação de um software começa bem antes do início do desenvolvimento. Desde as primeiras reuniões com clientes e levantamentos de requisitos, informações relevan-

tes para o desenvolvimento são produzidas. O volume de informações continua a crescer durante todo o ciclo de vida do software e a documentação deve acompanhar esse crescimento. Criar toda essa documentação demanda um esforço muito grande e também representa um grande custo no projeto, motivo pelo qual muitas vezes a documentação é mal feita. Tornar esse processo, em qualquer grau, mais penoso, certamente acarretaria a falha da plataforma em seu objetivo de aumentar a qualidade de documentação de um software. Temos então nosso primeiro requisito:

Requisito I: Ter um processo simples para a criação da documentação que, minimamente, não acarrete em mais esforço além do que já empregado para realizar essa tarefa.

Os documentos criados na fase de levantamento de requisitos ou modelagem diferem não só no foco ou grau de abstração, mas também em formato daqueles que são criados durante o desenvolvimento. Enquanto usamos diagramas de classe na modelagem, a documentação de código é puro texto. Ainda assim, tudo está relacionado. Um requisito leva à criação de determinadas classes que, por sua vez, têm seus métodos implementados, cada um, com sua própria documentação. Discutiremos mais adiante como essas relações serão mapeadas, mas é certo que temos aqui outro princípio arquitetural.

Requisito II: A plataforma deve suportar qualquer formato de documentação (ou tantos quanto possível), e ser capaz de estabelecer relações entre eles.

Além da documentação criada manualmente, muita informação pode ser extraída diretamente do código-fonte. Especialmente no caso de frameworks. As assinaturas dos métodos de um *hostspot* que classes estão disponíveis em um certo pacote, entre outras. Essa informação deve ser extraída automaticamente. Com o auxílio da documentação de código criada pelo desenvolvedor no próprio arquivo, como o padrão JavaDoc, por exemplo, essa extração pode se tornar ainda mais eficaz. O SoftDoc não pode, entretanto, se prender a uma linguagem específica, como Java, ou a um único padrão, como o JavaDoc.

Requisito III: O SoftDoc deve gerar automaticamente a documentação referente ao código fonte, suportando várias linguagens e seus respectivos padrões de documentação.

Dessa flexibilização surgem também oportunidades de aprimoramento. O padrão JavaDoc, por exemplo, possui um conjunto limitado de propriedades descritivas (e.g. @param, @returns). Utilizando-se do fato do SoftDoc ser flexível quanto ao padrão de documentação *inline*, podemos criar um padrão JavaDoc Estendido que inclua tags com @hotspot. Esta tag na documentação de uma classe, permitiria ao SoftDoc identificá-la como *hotspot* do framework e incluir esta informação na documentação gerada.

Mas ao longo do desenvolvimento do projeto, requisitos e, mais freqüentemente, códigos mudam. A documentação que foi gerada automaticamente a partir do código pode ser regerada facilmente para refletir essas mudanças. E quanto à documentação criada manualmente,

mas que está intimamente ligada ao código? É difícil identificar que documentos precisam ser alterados a fim de refletir essas mudanças. Mas é preciso manter a documentação em sintonia com o código. Cabe à plataforma proposta a solução desse problema.

Requisito IV: Identificar que documentação é afetada por uma determinada mudança no código. Atualizando automaticamente, quando possível, a documentação.

Em grandes projetos a documentação costuma contar com um enorme volume de documentos. Ainda assim, em softwares que são largamente utilizados, especialmente os *open source*, há um volume ainda maior espalhado pela *web*. O número de tutoriais, dicas e outros tipos de documentação, de boa qualidade, criados por usuários pode ser gigantesco. Não raro, muitas vezes encontramos a resposta à nossas dúvidas através de mecanismos de busca antes nessas que na documentação oficial. Se esses recursos são de tão grande ajuda aos usuários do software, eles certamente deveriam, ao menos, serem citados pela documentação oficial.

Requisito V: Permitir a referência manual de documentações de terceiros e, quando possível, identificá-las através de *trackback links* ou outras técnicas que permitam a citação automática dessas fontes na documentação oficial.

Se atingirmos nosso objetivo em atender aos requisitos mencionados acima, teremos uma vasta documentação sobre qualquer software que desenvolvamos. Poderemos certamente cobrir a maior parte das dúvidas de nossos usuários. Mas é preciso ainda garantir que a navegação pela documentação seja agradável e eficiente. Durante o uso da documentação, o leitor precisa freqüentemente subir e descer o nível de abstração a fim de entender um determinado recurso. Seguindo um tutorial ele pode, por exemplo, querer saber mais sobre uma classe. Ou, partindo de um modelo que demonstra os *hotspots* disponíveis verificar que métodos precisam ser sobrescritos ao instanciar um *hotspot* específico. Apesar da rígida divisão feita anteriormente, colocando usuários novatos e experientes em grupos distintos, é preciso enxergar que um usuário migra de um grupo para o outro ao longo do tempo, não com um salto, mas com uma transição gradual. Aos poucos o usuário aprofunda seu conhecimento sobre o software, um tema ou aspecto por vez, até que tenha coberto boa parte do software em profundidade.

Requisito VI: Encontrar relações e permitir a navegação entre diferentes níveis de abstração de documentação, permitindo que o leitor aprofunde-se no assunto que estuda no momento tanto quanto desejar.

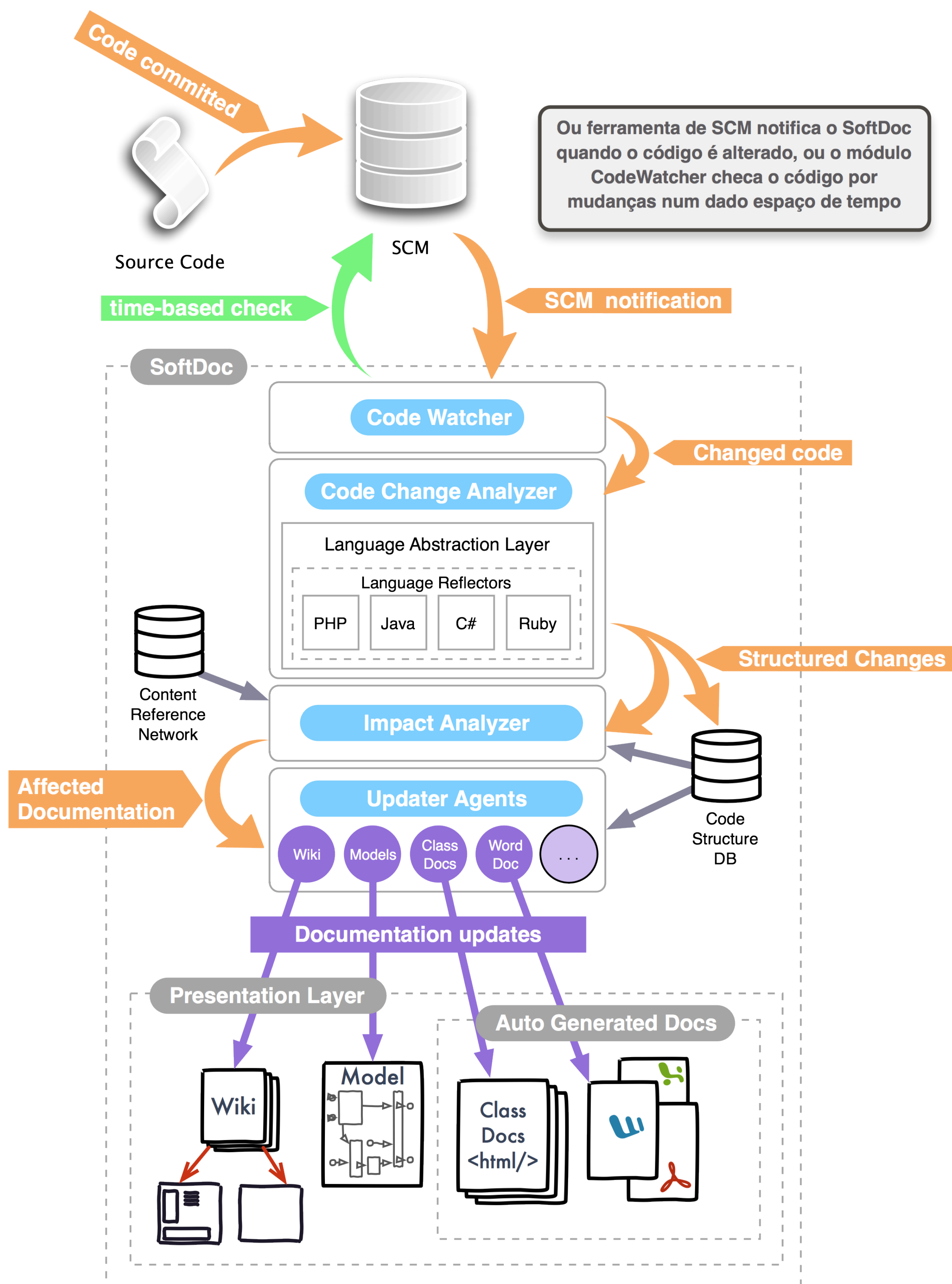
Com relação à usabilidade da documentação, cabe ainda uma última questão: como atender da forma mais eficiente cada leitor em seu público-alvo específico, quando temos tantos públicos-alvo? As tarefas executadas pelo administrador, pelo usuário e pelo programador de um sistema constituem conjuntos totalmente disjuntos. Obviamente, cada um deles necessita de um conjunto diferente de informações sobre o sistema para realizar essas tarefas. É ideal que se identifique que informação é relevante para cada agente e que a documentação disponi-

bilizada para cada destino tenha foco no conteúdo que é realmente útil a ele. Dependendo do perfil do usuário que está utilizando a documentação, alguns recursos tornam-se mais relevantes que outros. E isso deveria afetar a forma como a documentação lhe é apresentada. Como o atendimento ou não atendimento dessa questão é difícil de ser medido, não devemos defini-la como requisito, mas como uma meta a ser perseguida.

Meta: Adaptar-se ao perfil do usuário que está utilizando a documentação, fornecendo-lhe sempre as informações mais relevantes e no formato mais apropriado.

Arquitetura

Eis uma visão geral da arquitetura (o processo deve ser lido de cima para baixo):



Módulo Code Watcher

O módulo Code Watcher é responsável por monitorar as mudanças no código-fonte. Há duas maneiras de realizar essa tarefa. O caminho mais eficiente, mas nem sempre possível, é configurar a ferramenta de versionamento de código (SCM, do inglês *Source Code Management*) para avisar o módulo sempre que uma nova versão do código estiver disponível. Quando isso não for possível, o módulo poderá ser configurado para consultar a ferramenta de SCM a cada intervalo de tempo definido na configuração.

Esse módulo sabe como utilizar a ferramenta de SCM para obter apenas o código que foi alterado. Essas alterações são então encaminhadas para o módulo seguinte no diagrama.

Módulo Code Change Analyzer

O objetivo deste módulo é analisar as alterações no código-fonte recebidas e determinar se as mudanças podem ou não afetar a documentação. Isso acontece quando há mudanças na interface de classes ou na documentação presente nos próprios arquivos. Este módulo utiliza uma base de dados para armazenar a estrutura do código. Uma camada de abstração de linguagem permite que o módulo funcione de maneira agnóstica em relação à sintaxe da linguagem. Esta camada, por sua vez, conta com *parsers* que estruturam o código utilizando classes que permitem a obtenção de informações sobre o código parseado e que mantém a mesma interface, independentemente da linguagem.

Quando a modificação afeta a estrutura do código, inserindo, removendo ou alterando métodos ou classes, entre outras possíveis alterações estruturais, a estrutura do código é atualizada na base de dados e somente as alterações são encaminhadas, de forma estruturada, para o módulo seguinte.

Módulo Impact Analyzer

Vamos supor por um momento que possuímos um banco de dados em mãos que contém uma rede de conteúdo de onde podemos extrair informações sobre a relação entre recursos da documentação e o código fonte. Esta relação pode ser uma referência a uma classe em um exemplo de código, a um método em um tutorial, ou ainda a presença de uma classe num modelo. Mais a frente, veremos como esse banco é criado.

Com as alterações sofridas pelo código em mãos de forma estruturada e o banco de dados definido acima, podemos identificar que recursos podem ter se tornado obsoletos. Iterando todas as alterações e buscando no banco os recursos que citam ou descrevem essas classes e método s alterados, o módulo gera uma lista dos recursos que se tornaram obsoletos. Assim atingimos a primeira parte do requisito IV, identificar a documentação afeta por uma alteração

no código, a segunda parte deste requisito é cumprida pelo módulo seguinte, o módulo Updater Agents.

Módulo Updater Agents

A lista de documentação obsoleta pode ser bem diversificada, incluindo modelos UML, páginas html, documentos PDF, etc. Para lidar com tipos tão distintos, o SoftDoc utiliza agentes reativos. Cada agente é especializado em um formato de documentação. Eles são reativos porque só iniciam suas tarefas mediante uma comunicação do sistema de que algo precisa ser alterado ou verificado.

Esses agentes, além de alterar a documentação em que são especializados, trocam mensagens entre si para manter os documentos interconectados. Quando um agente percebe que sua documentação cita uma nova área do software, ele avisa aos outros agentes sobre seu novo conteúdo. Assim, se os outros agentes acreditarem que esta alteração é relevante e tiverem um conteúdo relacionado à ela, eles podem adicionar um *link* junto ao conteúdo relacionado, sobre o qual são responsáveis, para o novo conteúdo.

A estratégia de utiliza agentes especializados permite concentrar toda a responsabilidade de manipulação de um determinado tipo de documentação em um único elemento do SoftDoc. Essa modularização reduz o acoplamento da plataforma e facilita a expansão do suporte a novos formatos de documentação, atingindo assim nosso requisito II.

Os agentes responsáveis por documentações geradas automaticamente (e.g. a documentação gerada a partir do JavaDoc) terão também a responsabilidade de regerar essa documentação sempre que for necessário. Para cada linguagem ou padrão de documentação de código haverá um agente específico. Mais uma vez, essa estrutura facilita a expansão do suporte, agora para várias linguagens. Esse processo acontece de forma transparente e cumpre o estabelecido no requisito III.

Camada de apresentação

A camada de apresentação da documentação no SoftDoc é uma aplicação *web*. Assim, a documentação pode ser acessada tanto pelos desenvolvedores quanto pelos usuários se for colocada em um servidor público.

Muitos dos recursos disponíveis nessa camada podem ser encontrados numa wiki, como a MediaWiki (software *open source* que suporta a wikipedia.org). De fato, a própria MediaWiki pode ser utilizada para implementar esta camada, com algumas modificações. Vamos aos recursos que podemos encontrar na MediaWiki e seus respectivos papéis no SoftDoc:

- Fácil publicação

O conteúdo criado na wiki fica automaticamente disponível a todos os interessados. É possível ainda configurar o envio de notificações quando um novo conteúdo é criado. Usuários do software podem ainda optar por acompanhar um assunto específico e só serem notificados de atualizações na área que os interessa.

- Categorização do conteúdo

  Como dito anteriormente, certos formatos de documentação são mais apropriados que outros dependendo do nível de conhecimento do usuário sobre o software documentado. Com uso do recurso de *namespaces*, disponível na MediaWiki, é possível categorizar o conteúdo da documentação permitindo que o leitor busque pelo formato que mais lhe agrade.

- Internacionalização

  A tradução da documentação é indispensável para softwares de uso global. Mas a tradução sempre haverá um espaço de tempo entre a criação de um documento e sua tradução para outras línguas. Este recurso permite que usuário visualize um conteúdo traduzido para sua língua quando houver uma tradução disponível, caso contrário, a versão original é apresentada.

- Discussão

  A documentação do software é também um meio de comunicação, a medida que comunica, além do funcionamento do software, propósitos e intenções dos desenvolvedores. A wiki permite a expansão dessa face da documentação através da interatividade entre o leitor e a equipe. Através de comentários, os leitores podem sugerir melhorias, correções ou apontar partes do software não cobertas pela documentação.

- Colaboração e Versionamento

  Um dos recursos mais interessantes da wiki é a facilidade de colaboração para a criação de um único conteúdo. O versionamento desse conteúdo garante que qualquer equívoco, como a deleção inadvertida de um conteúdo, possa ser corrigido sem muito esforço.

- Tags

  Alguns formatos de conteúdo (e.g. vídeos e imagens) não são facilmente indexáveis, dada dificuldade de processar o seu conteúdo. Com uso de *tags*, ao ser publicado, esse conteúdo pode receber atributos semânticos que facilitarão a busca e o estabelecimento da sua relação com outros conteúdos.

- Acesso a conteúdo não visualizável

> Modelos de classe, diagramas de seqüência e outros tipos de conteúdo não podem ser visualizados em um navegador web em seu formato original. Podem, certamente, ser convertidos para imagens, mas perdem a capacidade de serem editados nos softwares onde foram originalmente criados. A wiki, permitindo a adição de qualquer arquivo, serve também como um distribuidor de conteúdo.

A extensibilidade da MediaWiki através de plugins, permitirá integrá-la ao SoftDoc de maneira transparente ao usuário.

Dois dos requisitos levantados são responsabilidade direta dessa camada:

Requisito I: Ter um processo simples para a criação da documentação que, minimamente, não acarrete em mais esforço além do que já empregado para realizar essa tarefa.

Requisito V: Permitir a referência manual de documentações de terceiros e, quando possível, identificá-las através de *trackback links* ou outras técnicas que permitam a citação automática dessas fontes na documentação oficial.

O primeiro requisito é cumprido pela própria MediWiki e pelos agentes do SoftDoc. A criação do conteúdo na wiki não requer o conhecimento de html, e, embora a linguagem utilizada pela MediaWiki para formatação do conteúdo seja simples, pode-se facilitar ainda mais essa tarefa pelo uso de um *plugin* de edição que fornece um editor de textos semelhante aos que usamos regularmente.

O requisito V é parcialmente obtido pela livre edição do conteúdo. A identificação automática de conteúdos produzidos por terceiros pode, inicialmente, utilizar a técnica de *trackback links*. Quando uma página da wiki é acessada através de um *link* em outro site, é possível obter o endereço da página que trouxe o visitante à wiki. Essa página então pode ter seu conteúdo processado pelos agentes do SoftDoc e, caso eles encontrem algum conteúdo relacionado à documentação, adicionarão *links* para essa página externa.

Essa técnica entretanto não leva em consideração a qualidade do conteúdo. Será preciso aprimorá-la no futuro com algum mecanismo de qualificação do conteúdo a fim de oferecer apenas recursos de qualidade aos leitores.

Updater Agentes + Camada de apresentação

No requisito V estabelecemos o seguinte objetivo:

Requisito VI: Encontrar relações e permitir a navegação entre diferentes níveis de abstração de documentação, permitindo que o leitor aprofunde-se no assunto que estuda no mento tanto quanto desejar.

Os agentes do SoftDoc são capazes de encontrar relações entre documentos independente do seu nível de abstração. Cabe à interface organizar essas relações de forma facilitar a navegação vertical do usuário. Embora os *links* para conteúdo externos sejam colocados ao fim de uma página, em uma seção específica, adotar essa estratégia para todos os *links* não seria uma boa abordagem. Ao invés disso, o *link* será colocado exatamente onde a classe, o módulo ou outro recurso for citado.

Discussão

Ainda é preciso ter disciplina durante o desenvolvimento do software. Afinal, a documentação precisa, em primeiro lugar, ser criada. Mas acredito que com o advento de uma plataforma que facilita esse processo torna-se mais fácil implantar essa cultura na equipe de desenvolvimento.

O SoftDoc não é uma plataforma completa, entretanto, também não é fechada. Há muitos pontos que podem ser melhorados, especialmente a identificação de relações entre documentos e entre a documentação e o código-fonte. Durante a pesquisa, tentei identificar esses pontos e arquitetar a plataforma para facilitar a implementação dessas melhorias.

Bibliography

Aguiar, A., G. David, et al. XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation.

Beck, K. (2001). Extreme Programming Explained, Addison Wesley.

Booch, G. (1994). "Designing an application framework." Dr. Dobb's Journal **19**(2).

Butler, G., R. K. Keller, et al. (2000). "A framework for framework documentation." ACM Comput. Surv. **32**(1es): 15-15.

Keller, R. K., J. Tessier, et al. (1998). "A pattern system for network management interfaces." Communications of the ACM **41**(9): 86-93.

Mili, H., H. Sahraoui, et al. (1997). Representing and querying reusable object frameworks. Symposium on Software Reusability. New York, ACM Press: 110-120.