DM

Environment Analysis
Ferramenta de geração de documentação a partir de um sistema SAP

DISSERTAÇÃO DE MESTRADO

Vitor Manuel Gavina Faria
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

fevereiro | 2020

Environment Analysis
Ferramenta de geração de documentação a partir de um sistema SAP

DISSERTAÇÃO DE MESTRADO

Vitor Manuel Gavina Faria
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Leonel Domingos Telo Nóbrega

Environment Analysis

Ferramenta de geração de documentação a partir de um sistema SAP

por

Vitor Manuel Gavina Faria

Tese de Mestrado em Engenharia de Software

Orientado por

Prof. Doutor Leonel Nóbrega

Faculdade das Ciências Exatas

Universidade da Madeira

2020

Agradecimentos

Ao meu falecido pai minha mãe e padrasto, pelo apoio dado em todas as fases da minha vida, e veio a se revelar crucial para a elaboração e concretização desta dissertação. Do fundo do coração agradeço-lhes a sua compreensão e a constante motivação com que pude contar ao longo deste trabalho.

À Ana Sofia Sousa e ao João Tiago França pela compreensão da minha ausência, em muitos momentos e paciência que tiveram comigo.

Ao meu orientador, Prof. Leonel Nóbrega, por todo o apoio e paciência que teve comigo. Agradeço-lhe pela sua boa disposição, compreensão, conselhos e total disponibilidade que revelou em todas as fases do trabalho.

Aos meus amigos, pela insistência e força que sempre me transmitiram para elaborar e finalizar o projeto, pela total compreensão da minha ausência, em muitos dos momentos de confraternização que tiveram lugar durante este período. E um especial obrigado, também, àqueles que apareceram na reta final e deram-me uma motivação extra.

Aos meus colegas de mestrado, que foram uma peça central.

Resumo

Com o crescimento dos projetos de software, em tamanho, a importância da documentação do código fonte tornou-se por demais evidente. Todavia, esta atividade por diversos fatores, como o cumprimento de prazos de entrega e o tempo gasto com o desviar da atenção dos programadores, vai sendo descurada, daí que a procura de ferramentas para tornarem este processo autónomo seja da maior relevância.

Efetivamente, um sistema bem documentado, será de fácil interpretação para qualquer um, tanto para aquele que entra em contacto pela primeira vez com o sistema, como para aquele que terá de realizar a sua manutenção.

A documentação de projetos de software é bastante abrangente pois inclui documentos referentes a requisitos, arquitetura, design, código fonte do software, testes, manutenção, e todos estes com informação essencial. Para este trabalho iremos nos focar apenas nas ferramentas para gerar documentação a partir do código fonte.

Este trabalho propõe-se a criar uma ferramenta de geração de documentação a partir de um sistema SAP[i], em linguagem de programação ABAP[ii], de forma a que se consiga produzir com pouco esforço documentação com qualidade, autonomamente.

Na solução apresentada foi implementado estruturas de suporte para a recolha da informação desejada, seguindo o paradigma da SAP e da linguagem ABAP. Para posteriormente, produzir a documentação, foi utilizado o Word, ferramenta de processamento de texto, tendo sido alcançados resultados bastante interessantes e aplicáveis. Assim, como resultado final obtivemos um documento Word com conteúdo pré-definido, pelo próprio utilizador da nossa ferramenta, discriminando todos os artefactos utilizados num dado programa tornando assim o código fonte mais percetível.

Palavras Chave: Documentação Software, Código Fonte, ABAP, SAP,

Abstract:

With the growth of software projects, in size, the importance of source code has become all too evident. However, this activity due to several factors, such as meeting delivery deadlines and the time spent deviating the programmers' attention, is being neglected, which is why the search for tools to make this process autonomous is great.

In fact, a well-documented system will be easy to interpret for anyone, both for those who come into contact with the system for the first time and for those who must carry out its maintenance.

The software design documents is quite comprehensive as it includes documents related to requirements, architecture, design, software source code, tests, maintenance, and all these with essential information. For this work, we will focus only on the tools to generate documents from source code.

This work proposes to create a document generation tool from an SAP system, in ABAP programming language, so that it is possible to produce with little effort with quality, independently.

In the solution presented, support structures were implemented to obtain the desired information, following the paradigm of SAP and the ABAP language. To study and produce documents, Word, a word processing tool, was used, and very applicable and applicable results were achieved. Thus, as a final result we obtained a Word document with pre-defined content, by the user of our tool, discriminating all the artifacts used in a given program, thus making the source code more noticeable.

Keywords: Software Documentation, Source Code, ABAP, SAP, OLE2Object

Conteúdo

Capítulo 1

1.1 Introdução

Consideramos documentação de software, como um artefacto cujo propósito é fornecer informação sobre o sistema a qual pertence [1]. Um dos grandes problemas, do desenvolvimento de projetos de software, é a falta de documentação, fazendo com que o código fonte seja incompreensível para novos programadores, ou, para responsáveis pela manutenção do software, constituindo um impedimento à compreensão do sistema.

Tal como todos os sistemas, os programas implementados na linguagem ABAP[1] do SAP[2] também possuem carências na área da documentação. Foi esta lacuna que procuramos preencher com este projeto, criando uma ferramenta capaz de analisar todo o ambiente de um dado pacote [2] e, a partir daí, criar documentação sobre o código fonte existente, permitindo a todos uma interpretação do mesmo de forma clara. A ferramenta criada foi designada por Environment_Analysis.

A documentação gerada pela Environment_Analysis é extraída diretamente do código. Por vezes, recorreu-se à análise sintática do mesmo código para poder obter o conteúdo, de forma a completar a informação apresentada na documentação gerada através do Word. Optou-se por este processador de texto da Microsoft por ser uma das ferramentas de produtividade mais comuns, utilizadas pela maioria dos utilizadores e empresas, saindo assim também do mais comum que seria na forma de um hipertexto HTML.

Neste capítulo são expressos os pontos de motivação, o contexto, objetivos, os resultados esperados e a estrutura deste documento.

[1] Linguagem de programação principal para o desenvolvimento de programas para o SAP
[2] SAP - Sistema integrado de gestão comercial transacional

1.2 Motivação

A área de geração de documentação para além de importante é muitas vezes descurada e não lhe é dada a devida importância por parte dos programadores, sendo estes uma das partes mais interessada em ter essa informação, ou porque vão dar continuidade a um dado sistema, ou porque irão realizar a manutenção do mesmo.

A ODKAS é uma empresa do sector dos serviços de implementação e consultoria SAP, logo sua atividade incide muito no desenvolvimento e manutenção de soluções em SAP. Daí ser de grande importância possuírem uma ferramenta que os auxilie na compreensão dos sistemas SAP, criando documentação de todo o código fonte existente nas suas soluções, resultando numa redução de custos e de tempo às suas equipas de desenvolvimento.

1.3 Contexto

Por parte da ODKAS foi-me facultado, através de um acesso VPN ao sistema SAP não havendo assim a necessidade de instalações ou configurações, um ambiente SAP próprio, onde teria a hipótese de trabalhar, descobrir a linguagem ABAP e o SAP, seus funcionamentos. Juntamente com o acesso, foi oferecida uma formação sobre conhecimentos básicos de SAP pela própria ODKAS. Depois do primeiro contacto com a tecnologia foi usada a documentação existente na internet [3] o que foi valioso durante todo o projeto pois a informação existente é vasta e mostrou-se fiável.

O SAP R/3 é um sistema integrado de gestão comercial transacional que se caracteriza, como todos os sistemas transacionais, pela alta taxa de atualização, grande volume de dados , podemos também considera-lo um sistema operacional integrado. O R/3, em específico, realiza o processamento de dados em tempo real, com uma arquitetura cliente-servidor de três camadas; base dados, servidor de aplicações e pela interface de cliente SAPGui .

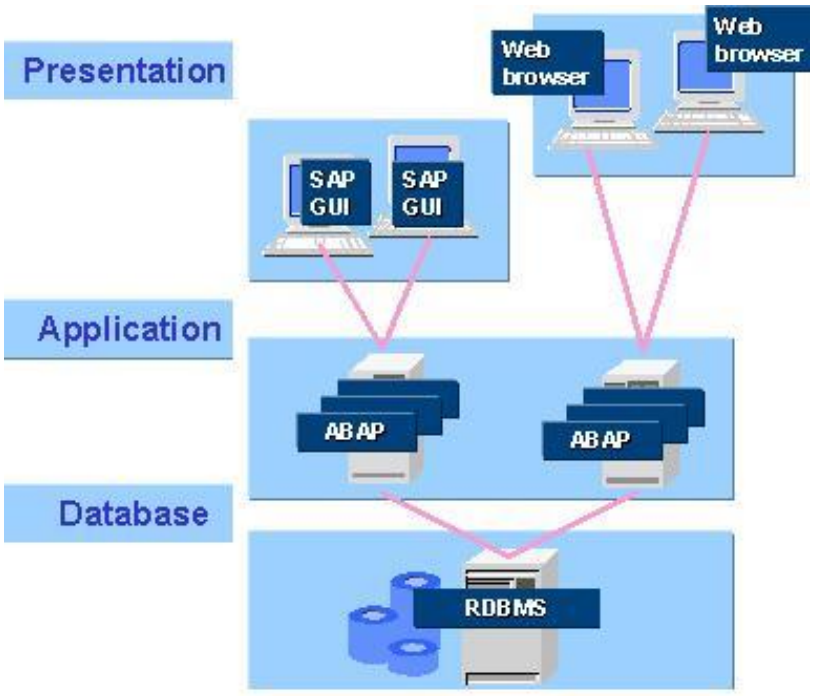


Figura 1 Arquitetura SAP R/3

1.4 Objetivos

O principal objetivo deste projeto é apresentar uma solução que consiga gerar documentação a partir de um dado pacote, nomeadamente, consiga percorrer os elementos existentes no pacote, *reports*, classes, funções, depois a partir do seu código-fonte detalha-lhos num documento Word. Identificando assim quais os *reports* (programas) existentes, descrevendo quais as funções utilizadas nestes, nome, linha onde é invocada, descrição e seus parâmetros. Pretende-se também que identifique as classes utilizadas, seus atributos, métodos e assinatura dos mesmos.

1.5 Resultados Esperados

Pretende-se com o estudo e investigação, perceber como a informação está presente no SAP R/3, como é estruturada e guardada, para assim conseguirmos da melhor forma criar nossas próprias estruturas, para guardar a informação recolhida. A informação no documento gerado terá de ser a que foi definida nos requisitos deste projeto, posteriormente enviaremos os resultados como output

num documento de Word. Esse output terá de ser conciso e claro, permitindo a sua fácil leitura e interpretação.

1.6 Abordagem

No SAP grande parte da informação de metadados é armazenada em tabelas próprias do sistema, que podem ser acedidas por consultas SQL. Num primeiro momento, optou-se por conhecer tais tabelas, pelo menos as de maior relevância, de forma a conseguir perceber qual a informação existente, a que iriamos necessitar, a forma de como relacionar e obter toda a informação pretendida e existente nessas tabelas. E por último, gerar o documento com o conteúdo pretendido.

De forma a conseguir recolher toda a informação desejada, são feitas várias consultas às tabelas com as condicionantes pretendidas. A informação em que não nos foi possível encontrar em tabelas do sistema, é feita uma análise sintática ao código-fonte de um dado programa, permitindo assim serem localizadas palavras chave e depois, completar, ter acesso aos dados pretendidos. De seguida, foi criar estruturas locais capazes de guardar a informação recolhida. Estas estruturas funcionam como um repositório de dados que vai sendo preenchido conforme a recolha de informação é feita. Depois desta estar concluída então com recurso à classe OLE2 e seus métodos é criado uma sessão de documento de *Word* onde passamos à escrita de toda a informação num documento de Word que irá ser apresentado como output ao utilizador.

1.7 Estrutura do Documento

O documento aqui produzido está estruturado em 5 capítulos, com os conceitos fundamentais ao tema proposto para este trabalho, à perspetiva pela qual fizemos a nossa abordagem de desenvolvimento.

No Capítulo 1, fizemos uma introdução ao tema do projeto, ao desenvolvimento da solução por nós apresentada. Tendo sido apresentados, a

motivação, o contexto, os objetivos, os resultados esperados e a nossa abordagem, finalizamos ainda com a apresentação da estrutura deste documento.

No Capítulo 2 são apresentadas noções mais relevantes sobre a temática da documentação de software, o que representa esta área seus desafios e boas praticas a seguir. Fazemos também uma descrição do estado da arte de algumas ferramentas e conceitos existentes na área de geração de documentação a partir do código-fonte do software.

No Capítulo 3, o foco é o sistema SAP R/3 que foi a base deste projeto, onde a solução proposta foi desenvolvida, é descrito com maior profundidade esta nossa abordagem e desenvolvimento da solução por nós apresentada para criação de documentação a partir do código-fonte.

No Capítulo 4, damos a conhecer a nossa solução

No Capítulo 5, realiza-se a avaliação e demonstração de resultados, com a devida discussão e conclusão.

Capítulo 2

2.1 Geração documentação de software

O termo "Geração de documentação" pode significar conceitos diferentes para pessoas com responsabilidades e funções diferentes, no processo de desenvolvimento de software. A documentação pode também ser direcionada ao utilizador final, de qualquer aplicação, para que este consiga ter uma visão formal de como funciona e do correto funcionamento da aplicação. A criação de documentação, num projeto de software começa antes do próprio desenvolvimento da solução, com documentação sobre os requisitos e arquitetura do software a desenvolver. Segue acompanhando o processo de desenvolvimento e a solução depois de concluída. A documentação de software auxilia os programadores, utilizadores sobre as rotinas e conteúdos da aplicação, facilitando a sua utilização o desenvolvimento de futuras evoluções ou a sua manutenção do software.

Vamos definir documentação de software ou do código-fonte, como um documento de texto escrito que acompanha o software a que diz respeito e o descreve.

Encontramos algumas linhas orientadoras nas normas IEEE 829 (ou padrão 829) para documentação de teste de software e, também, na norma IEEE 1074. Na norma IEE829, para desenvolvimento de processos de ciclo de vida de um software, de como deverá ser o formato da documentação, mas não estipula como todos devem e quais a serem produzidos, também não inclui critérios de conteúdo para esses documentos. Apesar de ser importante compreender e definir os conteúdos do documento de documentação de software, como por exemplo, é expresso nas normas acima referidas, primeiro achamos ser importante compreender os objetivos da documentação de software no seu todo.

Scott Ambler descreve a documentação de software como, "*qualquer artefacto externo ao código-fonte cujo propósito é transmitir informações de maneira persistente*" [4].

Neste trabalho, a nossa documentação será técnica e direcionada aos programadores, daí os conteúdos definidos para a documentação de software são de descrição do código-fonte. Tivemos em conta certos fatores importantes, para que a ferramenta por nós apresentada produzisse uma boa documentação: Clareza; Facilidade de acesso; Consistência; Estruturação.

A documentação apresentada deve ser clara e percetível ao publico alvo evitando assim más interpretações, deve ser evitado termos muito específicos e demasiada informação, para que esta não se torne difusa. Ser de fácil acesso, preferencialmente num formato, o mais universal possível, de forma a que todos consigam aceder à mesma sem pré-requisitos, ou constrangimentos que impossibilitem a consulta de forma rápida e concisa. Fazendo assim que a documentação gerada seja considerada uma ferramenta válida no processo de desenvolvimento e manutenção do software. Com esse objetivo consideramos como output um documento de Word devidamente estruturado onde através do painel de navegação o acesso à informação em todo o documento é simples e rápido.

A consistência é deveras importante porque permitirá que quem a consulte tenha uma única visão da realidade, fomentando assim a compreensão dos componentes do sistema, lembrando que estes podem ter informação recolhida de várias fontes. A consistência evitará também ter de navegar entre vários locais da documentação, procurando informação que poderá estar repetida, causando redundância.

O conteúdo apresentado de forma organizada é uma característica de uma boa documentação, logo a forma de como é estruturada a informação é da maior importância para a aceitação, ou não, por parte do leitor. A documentação desestruturada leva ao desinteresse e, por conseguinte, faz com que não seja aproveitada como suposto para o esclarecimento de quem a consulta.

As ferramentas de geração de documentação, são criadas de forma abstrata e por um programador ou equipa de programadores alheios ao software em desenvolvimento ou já desenvolvido. Existem, como já foi dito anteriormente, para colmatar a falta de documentação. É neste facto que existem, no nosso entender, dois problemas inerentes ao ser um processo que exclui os engenheiros de software do projeto em causa. Primeiro, e apesar de existirem modelos de documentação, estes apresentarem uma estrutura eficiente de forma a serem reutilizados, a informação necessariamente difere entre projetos de software logo as pré-estruturas da documentação ou modelos nem sempre apresentam a melhor solução para as necessidades exclusivas dos vários projetos de software. Segundo, de forma a sustentar a nossa opinião, recordemos Abdulaziz Jazzar e Walt Scacchi [5] que afirmaram "*muito do conhecimento sobre um sistema de software está na cabeça dos engenheiros de software*". Para complementar recordemos também a descrição feita por Alistair Cockburn sobre as dificuldades inerentes à comunicação "*a transferência intermediária de conhecimento provavelmente irá resultar em documentação de baixa qualidade*" [6].

2.2 Abordagens de Documentação

Donald Knuth nos anos 80 apresentou-nos uma técnica conhecida por *Literate Programming* [7], e tem como filosofia procurar ter como objetivo explicar aos humanos o que é pretendido que a máquina faça em vez de ter apenas o objetivo de dizer ao computador aquilo que ele deve fazer. Ou seja através de uma linguagem alfabetizada ir registando, a linha de pensamento inerente ao código fonte criado. Assim a técnica implica gerar um documento onde esteja tanto a documentação como o código fonte, estruturado mentalmente de forma a ser compreendido por humanos. Ora isto facilita a manutenção da consistência entre os dois tipos de conteúdo, e ainda assim permite que evoluam separadamente. A criação de programas utilizando a técnica *Literate Programming*, baseia-se na inserção de *tags* no princípio das linhas do código de forma a indicar, se o que se segue é documentação ou código, os blocos de código são explicados por qualquer ordem, mas normalmente um bloco de código é

explicado pelo texto que o precede. Com estas características esta abordagem revela-se um fomento à geração de documentação de qualidade, percetível a todos, mantém independência aos possíveis diferentes programadores e diminui-se também os esforços para perceber a arquitetura do sistema. Mas esta abordagem é pouco utilizada devido a uma serie de fatores destacando-se; A necessidade de combinar três linguagens em simultâneo, a do código fonte, a do documento final e a linguagem que permite a junção dos blocos de código e sua documentação; O código fonte torna-se demasiado complexo e longo o que dificulta a legibilidade e a sua compreensão; O esforço acrescido na programação de programas mais pequenos; Algumas incompatibilidades com os *debuggers* e os *refactoring*, pois o código fonte encontra-se com uma organização diferente à aquela espectada pelo compilador.

Outras abordagens alternativas são por exemplo, aquelas que as ferramentas Javadoc e Doxygen, descritas mais à frente neste documento, que aplicam a abordagem *single source*. Esta abordagem também segue o mesmo princípio de apenas um documento apresentar o código fonte e a documentação, assim as funções, métodos, classes, presentes no código fonte são documentadas através de uma sintaxe de comentários próprios pré-definida. O resultado é a geração de um documento normalmente no formato HTML.

Consideremos a abordagem *multiple source*, pois esta é aquela que a geração de documentação criada normalmente segue. A documentação, código fonte e todo o conteúdo interligado, ou, relacionado é mantido em separado e a sua combinação é realizada recorrendo à referenciação por localização (linhas), ou, copias dos conteúdos para o documento gerado pela documentação. Esta abordagem é dada a inconsistências se não for acautelado, este facto com um suporte eficaz por parte de ferramentas capazes de atualizar e gerir a referenciação existente.

Vamos também nos referir aqui à documentação baseada em XML (*Extensible Markup Language*), pois esta linguagem de anotação com a sua estrutura hierárquica, facilidade de navegação e extração de informação

juntamente à característica de ser *open-source* e facilmente extensível a novas funcionalidades. É uma abordagem bastante seguida pela geração de documentação dos nossos dias. Com as *tags* XML, a identificação do conteúdo do código fonte e a referenciação do código espalhado é facilitada aquando da geração de documentação, pode ainda ser transformado facilmente em outros formatos como HTML ou PDF, tornando assim a sua leitura ainda mais simples.

2.3 Ferramentas de geração de documentação

Nesta secção iremos descrever algumas das ferramentas existentes em outros sistemas que não o SAP, enquadrando assim o paradigma da geração de documentação a partir do código fonte e o contexto deste trabalho. De forma mais extensa detalhamos as ferramentas para linguagens de programação comuns e de uma forma mais sucinta as ferramentas para sistemas de bases de dados, pois este trabalho se incide em implementar uma ferramenta num desses sistemas.

2.3.1 Doxygen

Doxigen é um gerador de documentação criado por Dimitri van Heesch no ano de 1997. Standard para a linguagem de programação C++, mas também suporta outras linguagens tais como, o C, Objective-C, C#, PHP. Utilizando filtros é possível expandir a ferramenta de forma a suportar outras linguagens como, Java, PHP, Python, Perl, JavaScript, Assembly, VHDL e outras. O Doxigen, gera a documentação a partir do código fonte em formato HTML, RTF(MS-Word), PostScript, hyperlinked PDF e HTML comprimido. A documentação é extraída diretamente do código, permitindo que esta seja bastante consistente com o código fonte. Podemos ainda configurá-lo para a extração da estrutura do código fonte a partir de fontes não documentadas, o que é bastante útil pois assim podemos facilmente e de forma rápida aceder a partes do código não documentadas. O Doxygen, permite também visualizar as relações dos diferentes elementos através de gráficos das dependências, diagramas de heranças e de colaboração, são gerados automaticamente. É ainda possível criar simples documentação de forma por exemplo a gerar manuais de utilizador. O Doxygen,

é desenvolvido sob os sistemas operativos Mac OS X e Linux, mas tem uma portabilidade muito grande, como resultado funciona bem com outras versões do Unix e existem executáveis para Windows. Requer muito pouco esforço por parte de quem esta a criar a documentação. É suportado o Markdown[3] e, utilizando as *tags* HTML e/ou alguns comandos específicos do Doxygen, obtemos versões de documentação mais estruturadas. A indexação organiza e gera documentação navegável e com referências cruzadas, gera como *output* documentos XML estruturados, de fontes analisadas que podem ser utilizadas por ferramentas externas. Suporta a documentação de *namespaces, packages, classes, structs, unions, templates, variables, functions, typedefs, enums and defines*.

É compatível com JavaDoc (1.1), qdoc3 (parcialmente), e ECMA-334 (C#). Possui uma GUI[4] *frontend* (Doxywizard) de fácil edição permitindo executar o Doxygen de acordo com as opções escolhidas. A GUI está disponível nos sistemas operativos Windows, Linux e MacOSX. Gera automaticamente diagramas colaborativos e de classes em formato HTML, como mapas de imagens clicáveis e $\mbox{\LaTeX}$ (imagens encapsuladas de PostScript).

Utiliza a ferramenta *dot* do Graphviz[5] para a geração gráfica de dependências, *includes, calls*, estruturas de diretório, e representação gráfica das hierarquias das classes.

Permite o agrupamento de entidades, presentes nos módulos e cria a hierarquia dos módulos. Comentários são flexíveis e permitem colocar documentação adicional no cabeçalho do ficheiro antes da declaração de uma entidade. Gera uma lista de todos os membros de uma classe, incluindo os herdados, juntamente com o seu nível de proteção. Deteta automaticamente as secções public, protected and private, a extração dos membros das classes private são opcionais. Documenta funções, e variáveis globais, typedefs, defines e enumerations. Inclui um pré-processador completo de C, que permite uma análise apropriada de fragmentos de código e ainda a expansão de todas

[3] Markdown é uma linguagem simples de marcação originalmente criada por John Gruber e Aaron Swartz. Markdown converte texto em XHTML válido. https://pt.wikipedia.org/wiki/Markdown
[4] *Graphical User Interface* - https://pt.wikipedia.org/wiki/Interface_gr%C3%A1fica_do_utilizador
[5] http://www.graphviz.org/

as partes das definições de uma macro. Referencia desde a base à super class, heranças e membros overridden, existem mais de 100 opções configuráveis de forma a otimizar o output.

2.3.2 Headerdoc

Este é desenvolvido e mantido pela Apple Inc. É um conjunto de ferramentas para comentários embebidos em estruturas de código fonte, e cabeçalhos escritos em diversas linguagens o que subsequentemente produz um *output* bastante completo no formato HTML ou XML, a partir desses comentários. Os comentários são semelhantes, em aparência, aos do JavaDoc, mas tradicionalmente os do HeaderDoc fornecem um conjunto de *tags* mais formal, de forma a permitir um maior controlo sobre o comportamento do HeaderDoc. O HeaderDoc destina-se principalmente a ser usado com o OS X, como sendo uma parte integrante das OS X *Developer Tools*. No entanto tem sido utilizado em outros sistemas operativos, tais como, Linux, Solaris, e Mac OS 9. O HeaderDoc 8 suporta o JavaDoc *markup* e também uma ampla variedade de linguagens. Tem por *input* o código fonte especialmente comentado, gera documentação no formato HTML ou XML. A sintaxe das *tags* de comentário no HeaderDoc, são similares às do JavaDoc e também reconhece muitas das tags do Doxygen.

2.3.3 Javadoc

O Javadoc já vem incluído em algumas *IDEs* e faz parte do Develop tools. Suporta os sistemas operativos: Windows, OSX e Linux. É um gerador de documentação criado pela Sun Microsystems (1995) para documentar a API dos programas em Java, a partir do código-fonte, o resultado é expresso em HTML. As *tags* Javadoc nos comentários do documento, representam metadados sobre o código-fonte, ou seja, complementam com informações descritivas sobre a estrutura ou o conteúdo do código. Javadoc utiliza o sistema padrão de documentação de classes em Java, e muitas das IDEs desta linguagem já geram automaticamente um Javadoc em HTML. Fornece também uma API para a criação de *doclets* e *taglets*, que permitem a análise da estrutura de um programa

Java e é assim, por exemplo, que o JDiff[6] consegue gerar relatórios de alterações feitas entre duas versões de uma API. A sintaxe do Javadoc e do PHPDoc, já vêm sendo utilizados por um período longo daí terem sido já adotados por um número considerável de utilizadores, a sua estrutura de fácil manutenção também contribuiu para que tal acontecesse, por exemplo, cada função, seus parâmetros e retorno de valores são documentados ordenadamente utilizando *tags* especificas nos comentários.

2.3.5 PhpDocumentor

Joshua Eichorn no ano 2000 apresentou-nos o phpDocumentor, uma ferramenta que gera documentação diretamente do código fonte em linguagem PHP, tornou possível uma maior informação das funcionalidades embebidas no código. A documentação gerada tem como objetivo, ser complementar à informação convencional de forma a ser possível nas seguintes situações. Conjunto de bibliotecas, ou, aplicações que fornecem uma API, tais como o próprio phpDocumentor. Frameworks, como o Zend Framework[8] ou Symfony[9], arquiteturas que permitem plug-ins, como as do WordPress[10] ou PyroCMS[11]. O PhpDocumentor extrai e apresenta a informação em forma de gráficos e relatórios, diagramas de heranças, mostrando todas as subclasses e a implementação das interfaces. Ainda reporta; Erros que estejam no código fonte

[6] Javadoc doclet gera um relatório HTML de todos os pacotes, classes, construtores, métodos e campos que foram removidos, adicionados ou alterados de qualquer forma.
http://javadiff.sourceforge.net/

[8] O Zend Framework é uma coleção de pacotes PHP profissionais
https://framework.zend.com/

[9] Symfony Web Framework, escrito em PHP que segue o paradigma MVC.
https://symfony.com/

[10] WordPress Gestor de conteúdos para web, escrito em PHP com base de dados MySQL
https://wordpress.com/

[11] PyroCMS Gestor de conteúdos, escrito em PHP para o Laravel
https://pyrocms.com/

2.3.6 Ferramentas para a geração de documentação em bases de dados:

Sendo a solução que desenvolvemos para um sistema de base de dados, abordaremos de forma mais sucinta, neste subcapítulo, o tema da geração de documentação em bases de dados.

Nestes sistemas existem tabelas próprias onde o sistema guarda toda a informação referente aos objetos e elementos da base de dados em questão. Por exemplo no SQL server existem as tabelas sys.all_objects, onde está armazenada toda a informação de todos os objetos da base de dados, a sp_help, onde está armazenada toda a informação de todos os elementos da base de dados. No MySQL existem as tabelas *information_schema* que fornecem a metadata referente à base de dados, informações sobre o servidor MySQL, como o nome ou tabela, os tipos de dados de uma coluna ou privilégios de acesso.

As ferramentas de geração de documentação, existentes para recolha da metadata, realizam consultas a essas mesmas tabelas, com comandos SQL de forma a organizar e retirar a informação pretendida, existem vários que vão desde a versões não pagas a versões pagas, alguns deles são:

Dataedo – Suporta, SQL Server, Oracle, MySQL, MariaDB, Azure SQL Database, Amazon Aurora. Produz diagramas ER, separa os objetos em módulos. Tem um repositório global, para a documentação de todas as bases de dados e preparado para *Teamwork*. Exporta em formato HTML, PDF, MS Excel. A metadata é guardada num ficheiro ou repositório.

Redgate SQL Doc – Suporta, SQL Server. A integração no SQL Server é feita com o *Management Studio.* Exporta em formato HTML, PDF, MS Word, CHM. A metadata é guardada nas propriedades SQL Server**.**

2.4 Geração de documentação técnica em SAP

Como em qualquer outro ambiente de desenvolvimento de software, também, no sistema SAP a documentação revela-se de grande importância, de forma a que outros engenheiros de software consigam perceber o código fonte

criado, realizar a sua manutenção e evolução. Ao longo deste projeto, deparou-se que os programadores em SAP descuram bastante a documentação, chegamos a esta conclusão por observamos que o código fonte que fomos nos cruzando durante a investigação necessária ao projeto, continha poucos ou nenhuns comentários sobre o mesmo, ferramentas para documentação encontramos muito poucas e as encontradas foram de programadores independentes, que quiseram dar o seu contributo nesta área da documentação em SAP, e também não encontramos como sendo um requisito fundamental para todo o software criado em SAP. É certo que no SAP, cada função do sistema tem associada uma transação [8]. Associação essa identificada por um código que pode ter letras, números ou ambos, podemos observar na figura algumas das transações mais relevantes para este nosso contexto de documentação.

ABAP / Data Dictionary TC	
se11	Dictionary Definitions
se14	Database utility
se16	Data Browser (display only)
se16n	Modify :"&sap_edit" (uase16n)
sm30, sm31	Table views maintenance
se37	Function module editor
se38	Program editor
sa38	Program execution
se80	ABAP repository editor
se18	BAdI Definitions
se19	BAdI Implementations
se24	Class builder
smartforms	Smart forms administration
se43	Area Menu maintenance
se91	Message maintenance
se93	Transaction maintenance
scl	Code inspector

Figura 2 ABAP Dicionário de dados

Ao realizarmos uma simples consulta na transação SE11 - Dicionário ABAP, transação onde o programador consegue gerir todos os objetos do dicionário ABAP, como tabelas, tipos de dados, tipos de grupos, domínios. Temos acesso a grande parte da informação, por norma, contida na documentação.

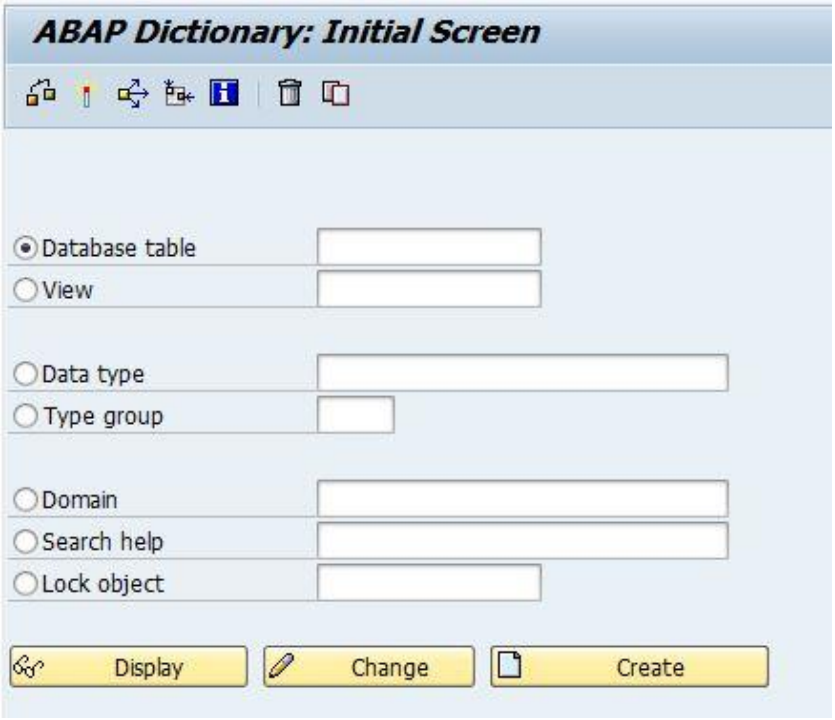


Figura 3 ABAP menu de consulta à transação SE11

Este paradigma e ambiente, leva-nos a concluir ser mais um entrave ao desenvolvimento da documentação de software no SAP. Na nossa investigação deparamo-nos, apesar de poucas, com algumas ferramentas de documentação, por opção escolhemos o gerador de documentação ZICDOC para apresentarmos como exemplo detalhado de uma ferramenta geradora de documentação a partir do código fonte de um programa em ABAP.

2.4.1 ZICDOC

ZIDOC é um programa feito em ABAP, com a função de analisar o código fonte de outros programas ABAP, gerando assim a documentação sobre este. A documentação gerada tem o formato HTML, e a extração da informação baseia-se nos comentários encontrados no código fonte analisado, identifica também

alguns objetos simples, como tabelas de dados, formulários, classes e blocos de código simples, tentando atribuir comentários a esses objetos. O resultado é um documento em HTML, onde é apresentado a descrição do programa analisado, o nível de detalhe depende muito dos comentários feitos pelo programador, que criou o código fonte analisado.

2.4.1.2 Aplicabilidade

O ZIDOC apresenta-nos um menu de iniciação, onde inserimos o nome do programa a ser analisado, as opções de analise e o nome a ser dado ao documento de HTML que irá ser gerado com a documentação referente ao programa.

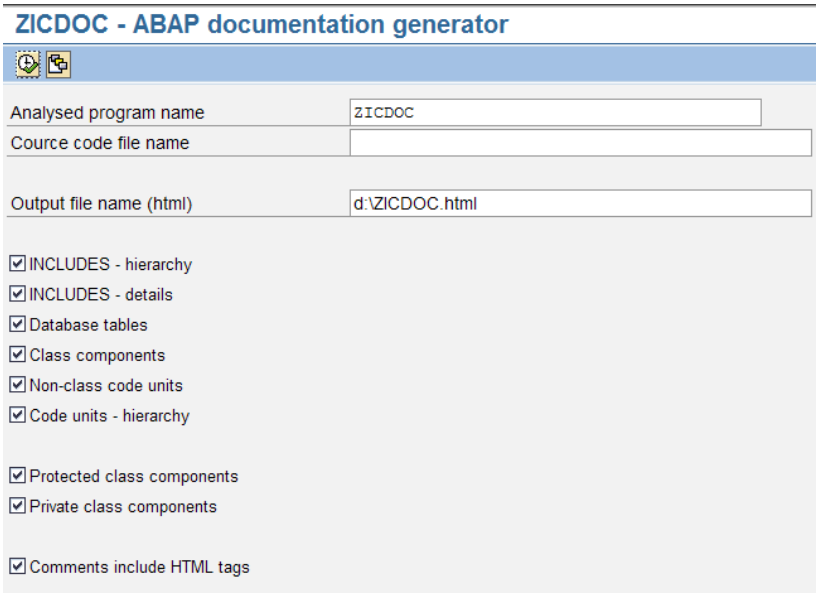


Figura 4 ZICDOC Menu de iniciação

Como já havíamos dito, este gerador de documentação baseia-se nos comentários existentes no código fonte, assim ao analisar o código fonte do programa a analisar vai atribuindo unidades de código aos programas, formulários, classes e outros objetos encontrados. Serão estas as informações a serem utilizadas para gerar a documentação para esses mesmos objetos, os tipos de declarações reconhecidas são:

- Comentários – São da maior importância existirem juntamente com o código fonte, através deles é feita a interpretação da utilização de funções, formulários, objetos e de alguns blocos de código individuais. O

Zicdoc lê os comentários, guarda-os como descrição dos objetos específicos, e segue certas regras estritas apresentamos algumas:

- As linhas de comentários são tratadas como uma string[9], contínua e são concatenadas numa secção ou segmento, o novo paragrafo começa quando é, encontrado uma linha em branco entre linhas de comentário e depois de qualquer bloco de código.
- É apenas atribuído um único paragrafo a cada objeto específico, o que significa, se tivermos múltiplos parágrafos separados por linhas brancas, apenas um será utilizado ou o primeiro ou o último.
- Todos os comentários que não comecem na primeira coluna por aspas (") e aqueles que separam blocos de código são ignorados.
- Zicdoc reconhece comentários especiais, como, o *& ou *", para isso têm de vir no inico das primeiras linhas do parágrafo de comentários. São tratados de forma especial, todos os outros seguintes são tratados de forma normal.
- Os comentários dentro de implementações de formulários, métodos e código fonte semelhante são ignorados.
- A colocação de tags pré-definidas, especiais também são colocadas no início do comentário juntamente com o caractere @ no início. Iniciam partes diferentes do comentário e são utilizadas para gerar formatos de output diferentes, como, utilizar cores ou tipos de letra diferentes.
- Incluem *tags* de formatação HTML, as típicas quebra de linha (<,>,&). São controlados pelos parâmetros no menu do programa Zicdoc.

- Identifica eventos como: START-OF-SELECTION, END-OF-SELECTION, INITIALIZATION, GET, AT.
- Pelas instruções de operações de base de dados como: TABLES, SELECT, UPDATE, INSERT, MODIFY, DELETE. É identificado as tabelas utilizadas no código fonte e assim a lista das mesmas é construída. Em

módulos de função invocados a partir do código fonte não são analisados.

- Classes e definições de interface, são criadas as listas de componentes.
- FORM e PERFORM, as listas de componentes das sub-rotinas são criadas.
- Definição de método como MÉTODO
- CALL, invocações de funções e métodos

Note-se que o programador organizou o Zicdoc da seguinte forma, três programas cada um com implementações e funções distintas; ZICDOC programa principal (main), é este que despoleta todo o desenrolar da geração de documentação; ZICDOC_CL_CODE_ANALYSIS, onde está definida implementação da classe para analise do código. A classe ZCL_IC_CODE_ANALYSIS, para a análise sintática do código e guardar os resultados em tabelas publicas. Faz também a leitura dessas mesmas tabelas e envia para ficheiro HTML; ZICDOC_CL_CODE_ANALYSIS_HTML, programa onde está implementada a classe ZCL_IC_CODE_ANALYSIS_HTML que lida com a análise feita ao programa em ABAP e criação do ficheiro HTML, com os resultados dessa análise. Juntamente o programador também definiu o aspeto e estilos desse documento num CSS (Cascade Style Sheets) a que deu o nome de ZIcDocStyle.css e este é obrigatório que esteja junto do ficheiro HTML.

2.4.1.3 Resultado

O resultado ou output deste gerador de documentação, começa com a hierarquia de programas e *includes*, ou seja, lista todos as inclusões existentes no programa e são apresentadas como sub-nós do programa analisado. Apresenta o nome, título, descrição e lista de eventos definidos no programa.

Seguem-se as tabelas utilizadas no programa com a seguinte informação, nome e título da tabela e tipo de acesso. No que tocas às classes apresenta o nome e sua declaração, atributos e métodos com comentários baseados na definição da classe.

O programador para atingir estes resultados para além de confiar na existência de comentários no código fonte a analisar criou um dicionário próprio e com as palavras chave em ABAP de forma a conseguir representar, identificar e interpretar o que vai acontecendo no código passamos a apresentar parte desse dicionário, apenas para constatarmos que é extenso e percebemos melhor como é feita a analise sintática do código fonte a analisar para posterior geração de documentação:

- @Info – descrição do objeto.
- @Author – autor do código
- @Date – data de criação
- @Change – detalhes de alterações
- @Param – parâmetros de um form ou método
- @Using – parâmetro de um form
- @Change, @Changing – parâmetro de um form
- @Import, @Importing – parâmetro de importação
- @Export, @Exporting – parâmetro de exportação
- @Return, @Returning – parâmetro de retorno
- @Receiving – parâmetro do metodo
- @Tables – parâmetro de uma tabela
- @Throws, @Exception, @Raising – excepções gerdas pelo código

Para os tipos de componentes:

- I - Interface
- T - Tipo
- C - Constante
- A - Aliases
- D - Data
- CD - Classe-data
- M - Método
- CM - Classe-método
- E - Evento

- CE - Classe-evento

Tipos de unidades de modularização de código e chamadas:

- F - Form
- FM - Function module
- CF - Customer function
- B - BADI
- T - Transação
- E– Evento principal (GET, INITIALIZATION, START-OF-SELECTION)
- CD – Definição de classe
- CI – Implementação da classe
- I – Definição de Interface
- M - Método
- S - Submeter de programa

E assim com recurso ao dicionário ao longo da análise os dados recolhidos vão sendo validados por estruturas de controle CASE.

```
CASE i_type.
    WHEN 'F'.
        e_desc = 'Form'.
    WHEN 'FM'.
        e_desc = 'Function module'.
```

Capítulo 3

ABAP

ABAP hoje é o acrônimo para *Advanced Business Application Programming*, mas inicialmente, a palavra ABAP era uma abreviatura das palavras alemãs Allgemeiner Berichts-Aufbereitungs-Prozessor [10] , que significam "*processador de criação de relatórios gerais*".

É uma linguagem de programação de alto nível que foi desenvolvida pela SAP. É utilizada na programação na camada do servidor de aplicativos da arquitetura do sistema SAP R/3. É também uma linguagem de quarta geração, ou, 4GL, linguagens de programação de alto-nível com objetivos específicos, como o desenvolvimento de software comercial de negócios, estas caracterizam-se por permitir ao programador especificar o que deve ser feito visando um resultado imediato. O termo 4GL foi usado pela primeira vez por James Martin no seu livro publicado em 1982 "*Applications Development Without Programmers*" para se referir a estas linguagens não-procedimentais e de alto-nível. A principal diferença entre as linguagens de terceira e quarta geração, é que estas primeiras são linguagens procedimentais que descrevem como fazer algo, enquanto a 4GL descreve o que queremos que seja feito. Uma 4GL que se popularizou foi a linguagem SQL (*Structured Query Language*), que se tornou um padrão para manipulação e consulta de bases de dados, sendo hoje em dia muito utilizada juntamente com linguagens de terceira geração.

O ABAP para além de imperativa e orientada a objetos, também é uma destas linguagens de quarta geração. Foi criada pela primeira vez na década de 1980, na época, o ABAP era utilizado como uma linguagem de relatório para a plataforma R/2 do SAP. Foi uma ferramenta que facilitou a organização, no desenvolvimento de aplicativos de negócios de mainframe para gestão financeira, gestão de materiais e contabilidade. A característica mais marcante e única do ABAP foi o fato de ter sido a primeira linguagem que utilizou o conceito de base de dados lógicos (LDBs). Isto permitiu que tivesse um nível de abstração

mais alto comparado com os outros que utilizam níveis mais básicos da base de dados.

A utilização do ABAP foi disponibilizada no desenvolvimento da plataforma R/3 do SAP. Esta deveria ser utilizada pelos clientes do SAP, com o objetivo de melhorar os aplicativos a desenvolver. Os clientes tiveram a capacidade de utilizar ABAP para o desenvolvimento de interfaces, bem como relatórios personalizados. É utilizada como a linguagem de programação principal para o desenvolvimento de programas para o SAP R/3 e foi com a continua evolução, do hardware e software de computadores, existente desde os anos 90 que promoveu a ABAP, e esta passa a ter uma ainda maior relevância na criação e desenvolvimento de sistemas e aplicações do SAP. Nos dias de hoje, tirando algumas mais básicas, quase todas as funções do SAP são desenvolvidas em ABAP. Em 1999 é lançada pela SAP a versão 4.6 do R/3 e juntamente a extensão ABAP Objects com a finalidade de aumentar ainda mais as funcionalidades desta linguagem. ABAP Objects, é um conjunto completo de instruções orientadas a objetos que foi introduzido na linguagem ABAP, esta extensão orientada a objetos do ABAP baseia-se na linguagem já existente e juntamente com as instruções introduzidas passa a ser possível a utilização de objetos ABAP em programas. Os objetos ABAP, suportam programação orientada a objetos, também conhecida como paradigma orientado a objetos, é um modelo de programação que une dados e funções em objetos. A restante sintaxe, destina-se principalmente à programação estruturada, onde os dados são armazenados de forma estruturada nas tabelas de base de dados e os programas orientados a funções acedem e trabalham com estes. O melhoramento do paradigma de orientação a objetos do ABAP, baseia-se nos modelos de Java e C ++. A implementação de elementos orientados a objeto no kernel da linguagem ABAP, faz com que os tempos de resposta ao trabalharmos com os mesmos aumentem consideravelmente.

O NetWeaver, é a atual plataforma de desenvolvimento do SAP, também dá suporte para o ABAP e Java. Fornece o IDE (Integrated development

Environment) para criar e gerir os programas e o código fonte em ABAP. O programador desenvolve e guarda, seu trabalho diretamente no servidor de aplicativos de onde são executados em tempo real.

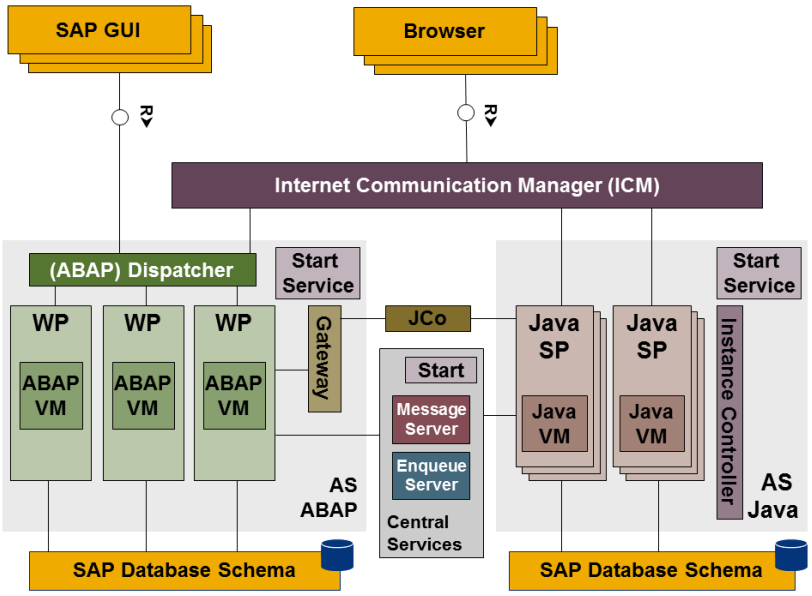


Figura 5 SAP NetWeaver Application Server

O *Workbench* do ABAP é, um conjunto de ferramentas utilizadas para desenvolver, testar e executar programas em ABAP e a versão mais recente também já inclui a ABAP Objects [11].

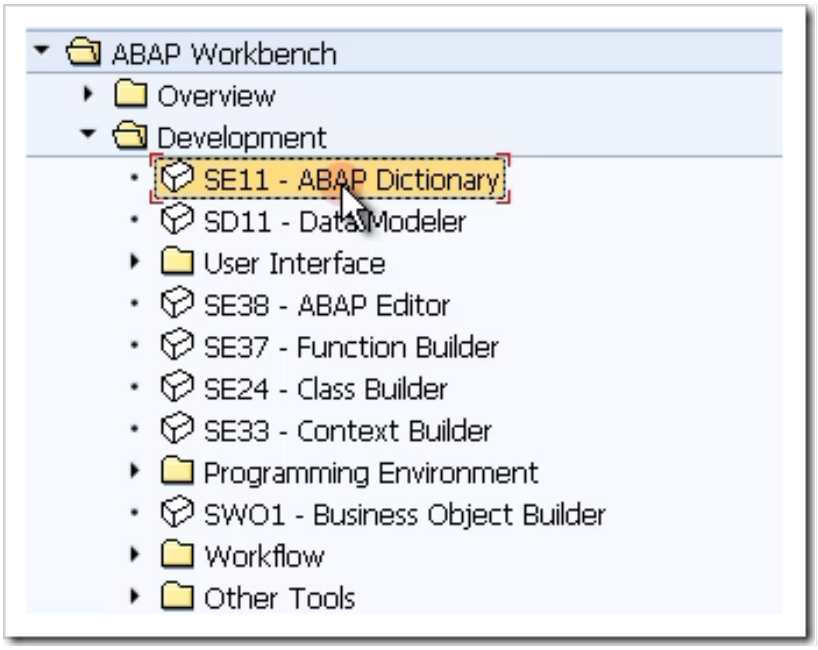


Figura 6 ABAP Workbench

O *ABAP Workbench* como podemos ver pela figura anterior é constituído por:

- *Repository browser* – Ferramenta que permite aceder, editar e gerir os objetos de desenvolvimento ABAP, tais como:
    - Pacotes
    - Programas
    - Function Group
    - Function Module
    - Classes Globais e Interfaces

Todos os objetos listados no *Repository browser* são apresentados com uma estrutura em arvore e podemos aceder cada objeto por meio do objeto pai ou vice-versa, conveniente quando não conhecemos o objeto de destino, mas conhecemos o objeto pai ou filho.

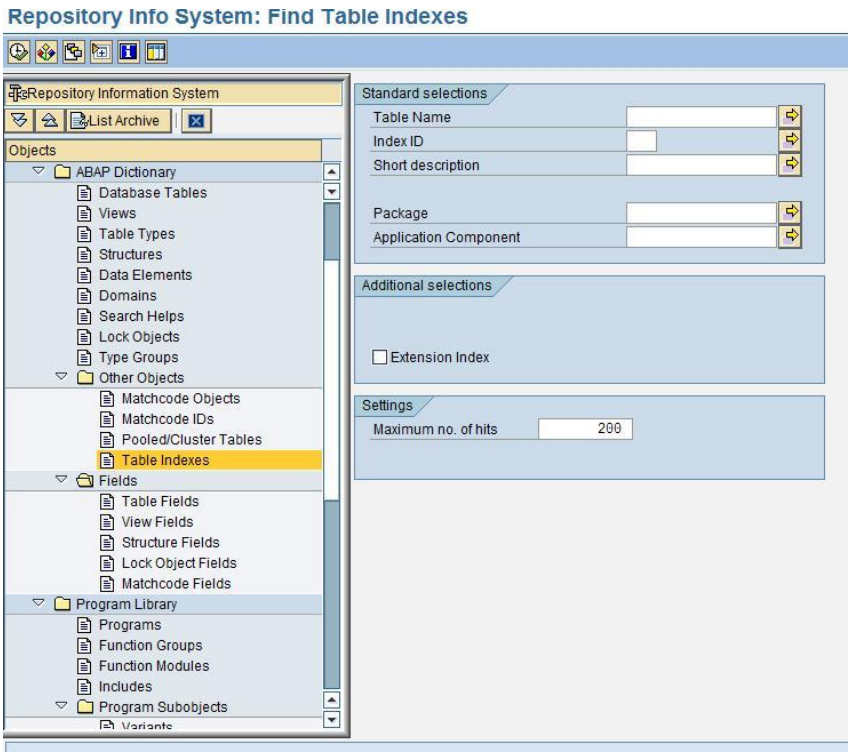
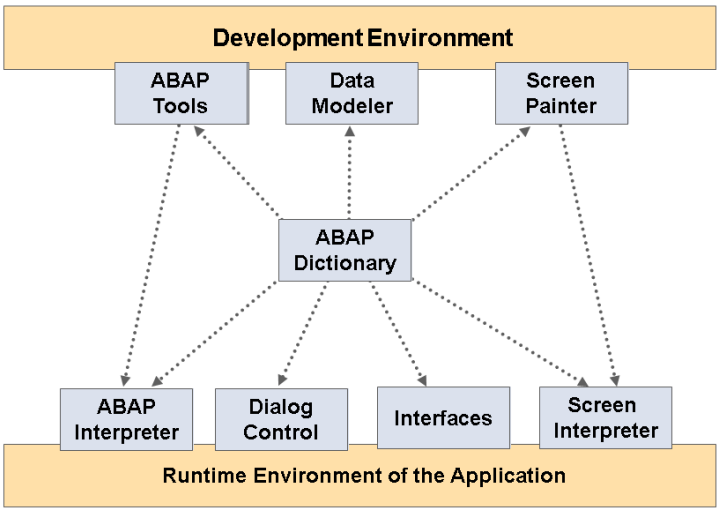


Figura 7 ABAP estrutura do repositório

- *ABAP Editor* – Ferramenta para editação de código fonte, projetada especificamente para lidar com as especificidades da sintaxe ABAP. Apresenta-se de três formas distintas;
    - Front-End Editor (modo código fonte)

    - Front-End Editor (modo plain text)
    - Back-End Editor (modo baseado em linha)
- *ABAP Dictionary* – Utilizado para criar e gerir a metadata, permite de forma centralizada uma descrição de todos os dados utilizados no sistema sem redundâncias. De forma a garantir a integridade, consistência e segurança dos dados toda a informação nova, ou modificada é disponibilizada automaticamente para todos os componentes do sistema.



- *Menu Painter* – Utilizado para desenvolver interfaces gráficas que incluem barras de menu e ferramentas.
- *Screen Painter* – Para manter componentes de monitor para programas online.
- *Function Builder* –Permite criar e manter grupos e funções e funtion modules.
- *Data Modeler* – Para suporte de modelação gráfica.
- *Workbench Organizer* – Mantém os vários projetos de desenvolvimento e gere a sua distribuição

SAP

Estávamos no princípio dos anos 1970, quando, Hasso Plattner e seus colegas Dietmar Hopp, Hans-Werner Hector, Klaus E. Tschira e Claus

Wellenreuther todos funcionários da IBM, tomam a decisão de começar a sua própria empresa, depois da IBM recusar a sugestão destes de desenvolver software desenhado para ser utilizado por vários utilizadores em simultâneo. Lançaram a sua empresa, com o nome SAP (*Systemanalyse und Programmentwickung* ou em inglês *System Analysis and Program Development*) e com sede em Mannheim, Alemanha. Enquanto todos os outros, mantinham esforços em desenhar vários produtos para ligar diferentes partes de um negócio em conjunto e com as capacidades necessárias, para desenvolverem um único sistema que iria unir todas funções de negócio de uma empresa. Este sistema de gestão única foi desenvolvido, com a capacidade de realizar contabilidade em tempo real e como um programa de processamento de transações, isto permitiu interligar toda a informação existente na base de dados, ficou conhecido como o R/1 sendo o R de *realtime processing,* significando que todos os dados eram processados imediatamente a partir da sua introdução no sistema. Hoje a SAP é líder mundial, no fornecimento de software corporativo e mais de 82.000 clientes em todo o mundo executam aplicativos SAP [12].

Como já referimos anteriormente o SAP, é um sistema cliente-servidor com uma arquitetura técnica de três camadas (*Three Tier Architecture*), estes sistemas derivam do modelo "*n*" camadas e caracterizam-se assim por ter sido retirada a camada, *application server* ou de negócio do lado do cliente, o desenvolvimento é mais demorado, mas em compensação o retorno é muito mais rápido e o controlo do crescimento do sistema é maior.

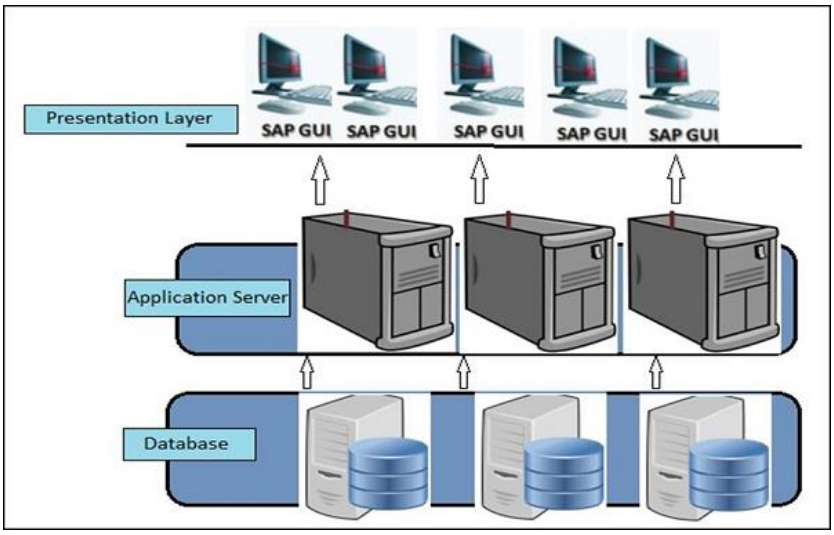


Figura 8 SAP Arquitetura de três camadas

Antes de abordarmos a arquitetura de três camadas do SAP, com maior profundidade, vamos antes nos referenciar ao *System Landscape* do SAP, que é, definido como sendo a arquitetura de servidores, a gestão e posicionamento dos servidores SAP, ou ainda, é uma coleção de instâncias SAP que têm uma relação de ciclo de desenvolvimento comum. O *System Landscape* é composto por todos os sistemas SAP instalados, pode ter vários grupos de sistemas que são ligados por rotas de transporte. Idealmente num ambiente SAP, será a existência de um cenário de três camadas (*Three Tier Landscape Architecture)*. Depois de avaliado, o número de clientes necessários, precisamos decidir como iremos distribuí-los entre os diferentes sistemas SAP, podemos configurar múltiplos clientes, independentemente uns dos outros num único sistema SAP. Contudo é importante nos lembrarmos ao configurar os dados que em clientes-cruzados configurações padrão e objetos do repositório são iguais para todos os clientes de um sistema único SAP, ou seja, as alterações realizadas num cliente aplicam-se automaticamente a todos os clientes do sistema. Sendo o *Three Tier Landscape Architecture,* a opção considerada como a ideal e mais comum, vamos abordá-la de forma a referenciá-la também como sendo a nossa opção ideal. Esta consiste em um sistema de desenvolvimento (DEV), um sistema de garantia de qualidade (QAS) e um sistema de produção (PRD). O fluxo de trabalho é unidirecional, tal como representamos na figura abaixo.

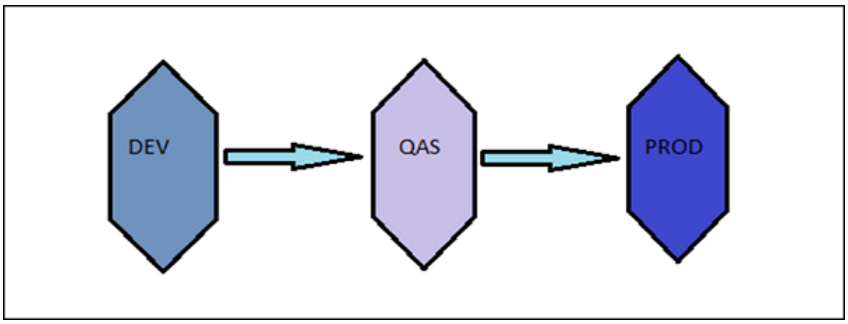


Figura 9 Fluxo de trabalho unidirecional

No servidor de desenvolvimento é onde, se realiza a criação de programas, é feita a configuração do sistema segundo os requisitos específicos da empresa e envia para o servidor de garantia de qualidade. Pode ser constituído por múltiplos clientes, por exemplo, 100 para realização de configurações e primeiras fases de implementação de um projeto, 120 para apenas desenvolvimento e 150 para alguns testes.

No servidor de garantia de qualidade, é feito os testes de desenvolvimento e configurações com vários parâmetros introduzidos pelos membros da equipa, antes de ser enviado para a o servidor de produção.

No servidor de produção são executadas as funcionalidades acedidas pelos utilizadores finais.

A arquitetura SAP, é definida como uma tecnologia *framework* do sistema SAP e pode ser alterada no decorrer do tempo ou com novo *software* ao contrário do *system landscape* que se mantém sempre igual.

Voltando à arquitetura SAP, para percebermos como o sistema SAP está organizado. No topo temos a camada de apresentação, que é constituída por qualquer aparelho de input utilizável para controlar um sistema SAP, pode ser um aparelho móvel, um sistema de utilização final ou SAP GUI e até mesmo por um acesso cliente através de um browser de internet. A verdade é que é nada mais do que um ficheiro chamado sapgui.exe, é instalado no posto de trabalho do utilizador, quando executado a interface SAPGUI, é mostrada e esta aceita o input realizado pelo utilizador enviando os pedidos para a camada de aplicação para ser processado, esta por sua vez envia o resultado para a SAPGUI, que por

fim formata o resultado de forma a ser apresentado ao utilizador. A camada de apresentação, comunica com a camada de aplicação executando assim todo o processamento, daí a camada de aplicação ser conhecida como o cérebro do sistema SAP pois é onde se dá todo o processamento.

A camada aplicação consiste em múltiplas instâncias e comunica com a camada base de dados. É um conjunto de executáveis, que coletivamente interpretam os programas em ABAP e gerem todo o input e output para estes, todos os executáveis iniciam ao mesmo tempo que o servidor de aplicações inicia e quando este termina todos os executáveis desligam-se. O número de processos a iniciar quando o servidor de aplicações inicia é, definido num ficheiro único chamado de perfil de servidor, cada camada de aplicação tem um perfil de servidor que especifica suas características, quando começa e enquanto é executada.

A camada inferior é a de base dados, é responsável por armazenar todos os dados e é mantida num servidor à parte por razões de performance e segurança. É um conjunto de executáveis que aceitam os pedidos vindos do servidor de aplicações, estes pedidos são enviados ao RDBMS (*Relation Database Management System*) que os envia para o servidor de base de dados e depois voltam a enviar a informação vinda da base de dados para o servidor de aplicações que por sua vez envia para os programas ABAP.

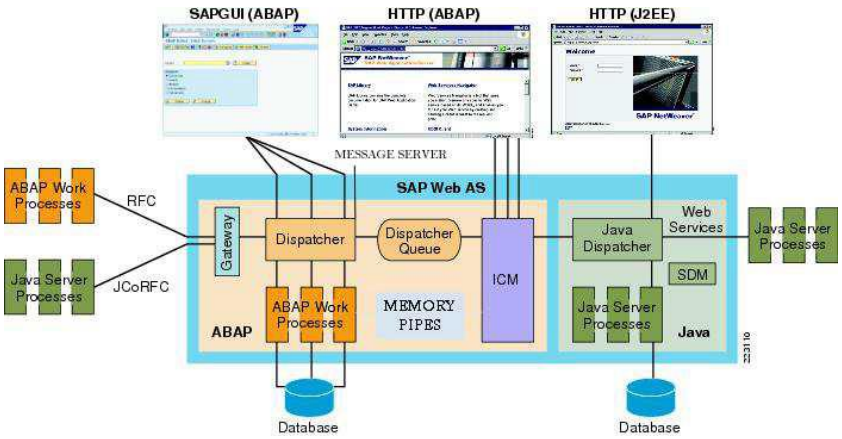


Figura 10 SAP -Componentes

Todos estes processos, que se desenrolam no sistema SAP, e a comunicação entre camadas baseiam-se em mensagens enviadas, expedidores (*dispatcher*) e filas, e o seu bom funcionamento deve-se a certos componentes chave existentes no sistema tais como:

- Servidor de Mensagens – lida com a comunicação entre os vários *dispatchers* distribuídos no sistema ABAP.
- *Dispatcher* (expedidor) – Distribui os pedidos para os processos de trabalho (*work processes)*.
- Fila de *Dispatcher* – Os vários tipos de processos são guardados nesta fila
- Gateway – Ativa e permite as comunicações num Sistema SAP e entre um sistema SAP e sistemas externos.
- ABAP-Processos de trabalho – Executa separadamente as etapas de diálogo dos aplicativos R/3. Os tipos de trabalho podem ser
    - Diálogo – Responsável por processos de diálogo
    - Atualização – Responsável pelas atualizações
    - Atualização2 – Responsável pelas atualizações consideradas menos críticas
    - Background – Responsável por trabalhos de background
    - Spool – Responsável pelos pedidos de output
    - Enqueue - Responsável pelos bloqueios
- *Memory pipes* – Ativa a comunicação entre o ICM e o ABAP-Processos de trabalho
- *Enqueue Server* – Lida com os bloqueios lógicos que são definidos pelas aplicações executadas em Java num processo do servidor.
- Serviços Centrais – Cluster em Java, requer uma instância especial (grupo de recursos, como, memoria processos de trabalho) nos serviços centrais para gerir bloqueios e transmitir mensagens e dados. É um conjunto de processos que trabalham em conjunto para conferir fiabilidade ao sistema.
- *Java Dispatcher* – Recebe os pedidos do cliente e remete-os para o processo do servidor

- SDM (*Software Deployment Manager*) – Utilizado para instalação de componentes J2EE.
- *Java Server Processes* – Consegue processar um grande número de pedidos em simultâneo.
- *Threading* – Múltiplos processos executados separadamente em segundo plano, a este conceito dá-se o nome de *Threading*
- *ICM* – Permite comunicação entre o sistema SAP e os protocolos HTTP, HTTPS e SMTP, permitindo assim aceder ao sistema SAP através de um simples browser bastando para isso colocar o devido URL.

A maior parte destes componentes, encontram-se na camada de apresentação. Num sistema SAP, existem várias instâncias do servidor de aplicações e apesar do número de instâncias de diálogo que o utilizador execute existe apenas uma instância central, que contém a mensagem de servidor.

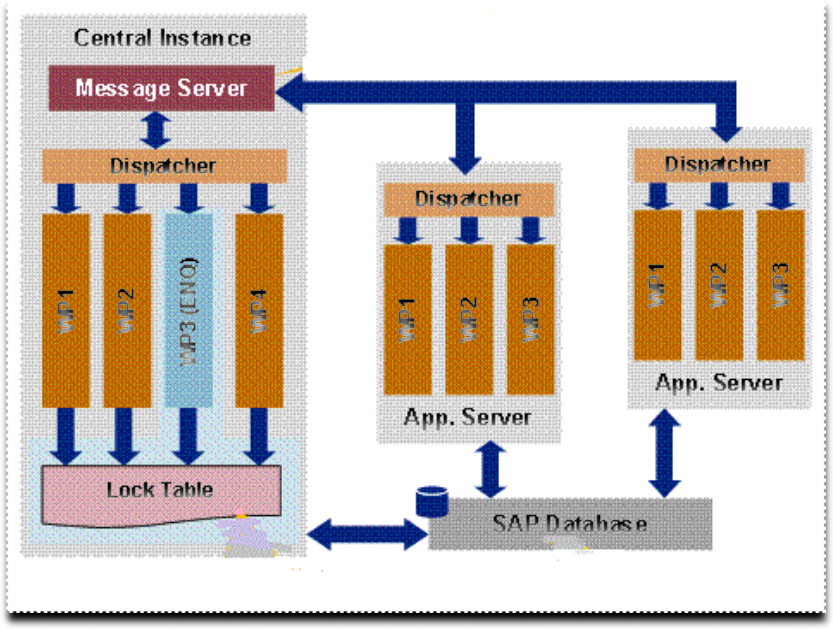


Figura 11 SAP Instâncias do servidor

Como já referimos anteriormente, a camada de aplicação consiste em múltiplas instâncias e comunica com a camada base de dados, e é neste contexto que a instância diálogo é da maior relevância, pois é, com a utilização desta que tudo acontece. A instância diálogo, consiste nos componentes *ICM*, *Dispatcher*, *Work Processes*, SAP *Gateway* e o *Message Server*.

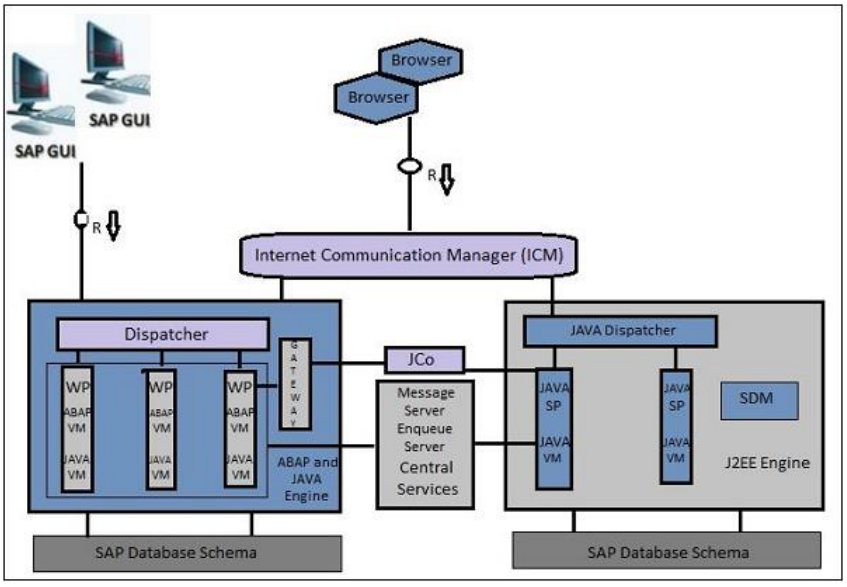


Figura 12 Instância de diálogo

Importa voltarmos a nos referir ao *Work Processes,* pois é neste ambiente que os programas são executados. Num sistema SAP, existem vários tipos de programas, aqui neste trabalho vamos apenas referenciar três dos mais utilizados:

- Programa Executável
- Module Poll/Online
- Grupo de Funções

O Programa executável, ou Report, são programas utilizados para fazer relatórios de apresentação de dados ou executar tarefas em segundo plano. São os únicos que podem ser invocados, através do comando *submit* e quando executados passam por eventos como o *initialization, start-of-selection, form* e outros. Foi com este tipo de programa, que criamos a solução apresentada aqui neste trabalho e com estes que a maioria dos programadores ABAP trabalham.

Os *Module Poll/Online*, são semelhantes aos Reports, mas não têm os eventos e não podem ser invocados pelo *submit*. São mais utilizados na interação com o utilizador e podem ser invocados através da transação associada.

Grupo de funções, apesar de serem considerados um programa, são conjuntos de funções. São utilizados, para a criação de código que possa ser reutilizável por outros programas e são os únicos que podem conter Modulo de funções.

Todos os programas, executados num sistema SAP são executados no *Work Processes* que por sua vez é executado na camada aplicação, funcionam independentemente do sistema operacional, do posto de trabalho do utilizador e da base de dados com a qual interagem. É o *Work Processes,* que executa os programas e que tem acesso às áreas de memoria onde residem os dados e objetos, que o programa utiliza.

É o *Dispatcher* que controla a distribuição no *Work Processes*, tendo conhecimento de quantos processos estão disponíveis quando um utilizador aciona uma transação, fornece a esse utilizador um processo de trabalho. Otimiza o máximo possível de forma a que o mesmo processo de trabalho siga as etapas sequenciais de diálogo de um programa.

Elemento também importante é o processador ABAP, responsável pelo processamento logico dos programas. Para além das operações logicas e cálculos aritméticos verifica as permissões de escrita e leitura na base de dados através da interface base de dados.

Programa SAP ABAP

Como se constata, pelo que foi descrito neste capítulo, SAP é um sistema com uma arquitetura técnica própria (Apresentação, Aplicação, Dados), ou, numa visão menos técnica é um produto. E ABAP, uma linguagem de alto nível de quarta geração e também a linguagem de programação principal no ambiente SAP.

A IDE, para o programador e ambiente de execução estão no servidor SAP, todo o código fonte criado é guardado na base de dados do sistema SAP, assim na máquina do utilizador e do programador não está nada. Num programa com código fonte em ABAP, quando este é executado pela primeira vez, ao ser

compilado gera um código intermediário conhecido como *ABAP Load*, a máquina virtual do sistema SAP interpreta esse código e converte-o na linguagem da máquina especifica, processo semelhante ao existente na linguagem Java. Como ABAP é uma linguagem 4GL, os comandos e a sua sintaxe têm um nível alto muito próximos da linguagem humana, são adotados alguns padrões de nomenclatura facilitando assim o desenvolvimento de código fonte. Por exemplo, tudo aquilo criado fora do sistema SAP, seja tabela, classe, programa deverá ter na sua identificação a letra Z ou Y na primeira letra do seu nome. Neste contexto SAP ABAP, o que proporciona a execução de um programa é a transação, que funciona como um simples atalho, todo o programa que passe a fazer parte do sistema SAP tem necessariamente de ter associado a si um código da respetiva transação. O SAP, já possui código fonte próprio e este não pode ser alterado, de forma a torná-lo mais flexível existem os *enhancements points*, permitindo dentro destes blocos de código inseridos no código fonte padrão do SAP contruir código fonte específico para necessidades especificas sem ter de criar programas inteiros, de forma a colmatar essas especificidades.

Um programa em ABAP, como em outras tantas linguagens, por norma é estruturado em duas partes, a secção de declaração e a de processamento de blocos. Na secção de declaração, é onde definimos, os tipos de dados, estruturas, tabelas, variáveis e campos individuais, a serem utilizados pelo programa. Na criação de programas em ABAP, não só declaramos variáveis globais, mas também nos é possível declarar variáveis especificas para segmentos específicos do código fonte do programa. É nesta secção que definimos os parâmetros de *selection-screen* para os *Reports* (programas).

De seguida, temos a secção de processamento de blocos, onde toda a lógica para o programa será escrita. Esta secção definida dentro dos programas, poderá ser invocada pelo processador *Dynpro,* dependendo das regras especificas criadas dentro do programa, são normalmente programação lógica que permite o encapsulamento do código fonte.

O que será comum a todos os programas é que irão ler e escrever dados na, e da, base de dados do sistema SAP. No *ABAP Workbench,* como já vimos anteriormente é onde se encontram todos os componentes necessários ao desenvolvimento de um programa ABAP, existe a transação SE11 para executar o *ABAP Dictionary,* e também a transação SE38 associada ao *ABAP Editor* onde todo o código é criado. Foi este componente, o *ABAP Editor,* o mais utilizado ao longo deste projeto na criação da solução apresentada por nós.

Capítulo 4

Requisitos

Foi-nos proposto, no seguimento deste trabalho, criar um programa SAP para gerar documentação de forma automática, com base num input. Uma solução capaz de gerar documentação, para anexar a uma transação SAP, num servidor central. E com a capacidade de apresentação deste documento anexo à logica de transação SAP, ou seja, a visualização do documento com chamada à interligação criada em SAP.

Foram definidos como requisitos funcionais:

- Permitir analisar um package ou programa individual.
- Identificar as funções de cada programa e seus parâmetros.
- Identificar as classes utilizadas e descrever seus métodos, atributos e dependências (sub classes e super classes)
- Gerar um documento com toda a informação

Como requisitos não funcionais foram definidos:

- Permitir uma utilização intuitiva e fácil
- Permitir configurar um scope da informação pretendida no documento gerado
- Documento gerado de fácil consulta

Environment Analysis

Como ficou demonstrado, ao longo deste documento, a documentação de um código-fonte de qualquer programa, é da máxima importância não só para a sua manutenção como também para uma posterior ampliação. Existem inúmeras ferramentas para gerar documentação, mas os programadores não as utilizam frequentemente. Devido à modificação dos programas, passado algum tempo a documentação gerada fica desatualizada, a complexidade e extensão dessa

mesma documentação gerada tornam a utilização de ferramentas de geração de documentação pouco atrativa. A nossa visão é de que uma ferramenta de geração de documentação, claro, conte com as informações adicionais no código-fonte, como os comentários, mas que dê a sua maior atenção e relevância às informações que o código-fonte em si têm e podem-nos dar. Mais ainda assim não produza demasiada informação e que esteja apta a receber sugestões de implementações novas, por parte dos programadores alvo da sua utilização. Se falharmos em gerar uma boa documentação, estaremos também a falhar na facultação de informações uteis sobre a criação do programa a analisar, e esse desenvolvimento torna-se então num candidato a se tornar num *monstro Z* [12]sem utilidade. Idealmente, seria apresentar uma ferramenta que criasse automaticamente documentação apenas a partir da informação existente no código-fonte, e que gerasse documentação com todo o conteúdo pretendido e de forma rápida de modo a ser utilizada sempre que existisse alguma alteração no código-fonte sem prejuízo do custo tempo e esforço. A ferramenta por nós idealizada, e aqui apresentada foca-se essencialmente na informação que podemos adquirir a partir do código-fonte sem recorrer aos comentários, mas foi implementada de forma a que se necessário e desejado pelo programador, ser fácil a introdução de um dicionário a ser construído em conjunto com a equipa de desenvolvimento para que passe a interpretar os comentários que possam existir no código-fonte. O *Environment Analysis*, foi este o nome que demos à, solução apresentada por nós aqui neste projeto, surge na sequência dos conceitos desenvolvidos ao longo deste documento que foram aplicados a um domínio concreto no âmbito de uma organização real. Pretende ser, uma ferramenta de geração de documentação rápida e pratica a gerar documentação, com conteúdo conciso e aberta a novos desenvolvimentos consoante as necessidades futuras. Tivemos em conta na implementação da solução a sua usabilidade, ou seja, foi pensada tendo em conta o utilizador final e as suas expetativas e procurou-se algo intuitivo, claro e simples de utilizar para não contribuirmos para o descuramento verificado no

[12] Tudo aquilo criado fora do sistema SAP, seja tabela, classe, programa deverá ter na sua identificação a letra Z ou Y na primeira letra do seu nome.

processo de criação de documentação de software. O desempenho, todas as operações envolvidas na solução proposta procuram ser efetuadas no menor tempo possível. E a consistência de informação entre os dados retirados do código-fonte e do conteúdo dos documentos produzidos.

Visão Geral

Idealizamos a abordagem ilustrada pela figura 13

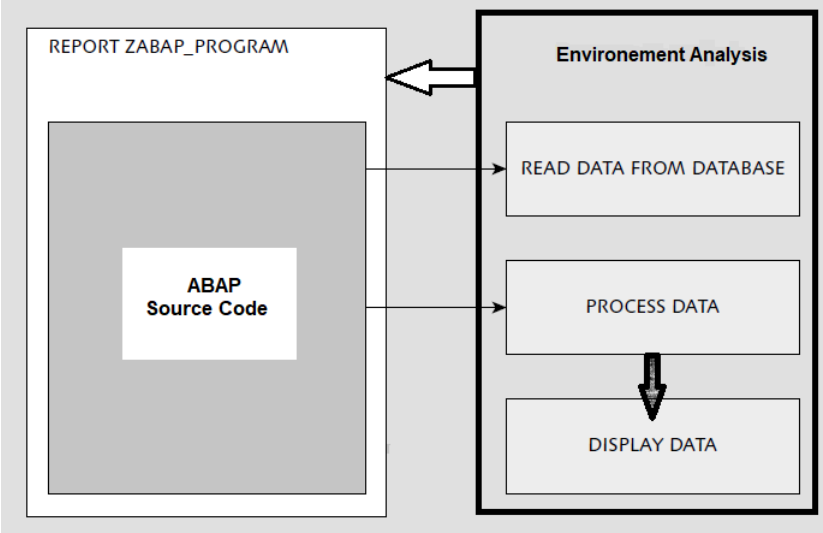


Figura 13 Environement Analysis

Na solução destacamos ainda a opção de escrita no documento, em tabelas ou em texto normal. E o utilizador, poder configurar os conteúdos pretendidos no documento final, dentro um leque de opções conforme podemos observar na imagem seguinte do menu inicial.

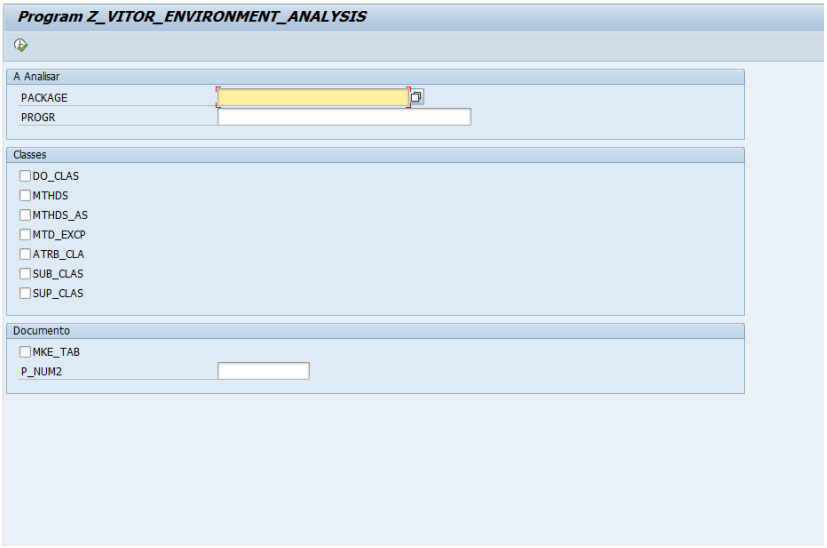


Figura 14 Menu Inicial

Estrutura

Para implementarmos a abordagem por nós idealizada e de encontro aos conceitos desenvolvidos neste projeto, optamos por criar estruturas internas capazes de guardar a informação desejada e obtida a partir do código-fonte, permitindo-nos assim também um maior controlo e manuseamento dessa mesma informação. Assim, foram criadas estruturas internas de forma a guardar a informação relativa a:

- Programas
- Funções
- Parâmetros das funções
- Classes
- *Parsing* do código-fonte

A nossa abordagem passou por recorrermos a artefactos como, rotinas, sub-rotinas, classes pré-existentes e algumas criadas por nós para atingirmos nossos objetivos, a opção que tínhamos idealizado era a de recorrermos apenas ao paradigma de programação orientada a objetos. Mas após melhor conhecimento da linguagem ABAP e do SAP, e sendo este projeto um projeto com pouco alcance ao nível de equipas de desenvolvimento desta temática, o custo esforço e recursos a serem utilizados eram muito maiores do que aquele se não

utilizássemos os recursos e potencialidades do SAP/ABAP, assim optamos também por recorrer ao paradigma da programação imperativa.

Programas

Na hierarquia dos conteúdos gerados, programas é o que está no topo. A informação sobre os programas é guardada em diversas tabelas internas do SAP. Optamos pelas tabelas TADIR e TRDIR, pois ao fazermos a união das duas a intersecção resultante dá-nos os dados que pretendemos.

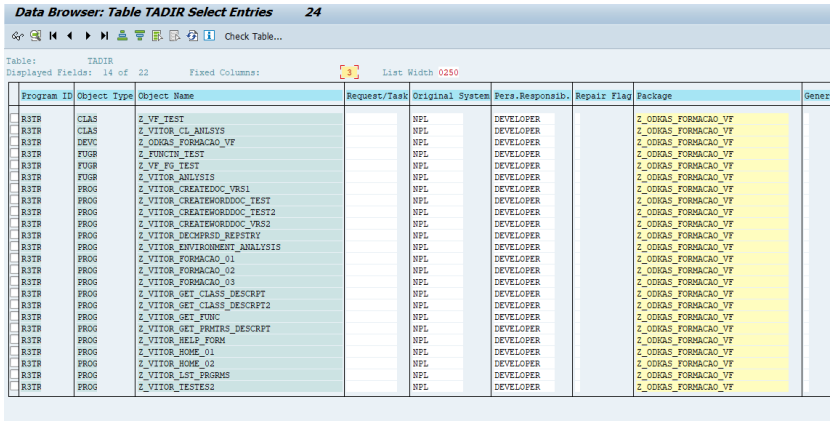


Figura 16 Tabela TADIR

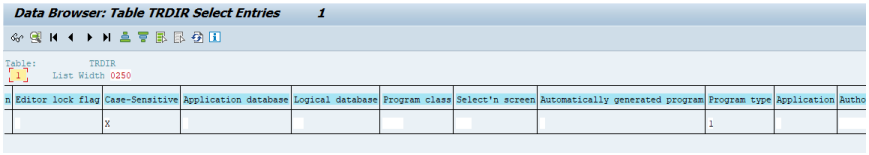


Figura 15 Tabela TRDIR

E o resultado, são todos programas existentes num determinado pacote que sejam executáveis. Esta implementação está no método "*get_program_list*" da classe "*z_vitor_cl_anlsys*". Assim ficamos com a estrutura interna, pré-definida na classe, "*prg_tab*" preenchida, e é daqui o nosso ponto de partida para a análise a ser feita ao código-fonte de todos os programas (*Reports*), existentes num dado *Package*. Claro, se optarmos pela geração de documentação de um programa apenas, o Environment Analysis irá se focar apenas nesse programa. Nunca poderemos analisar os dois, mas sim, temos de optar ou por analisar um pacote ou um programa individualmente.

Funções

Partindo dos dados recolhidos na tabela "*prg_tab*" e recorrendo à um *Loop* vamos percorrer a tabela e retirar o nome de todos os programas existentes no pacote em questão que esta a ser analisado.

```
158
159   START-OF-SELECTION  .
160     LOOP AT prg_tab INTO nome.
161       program = nome-nome_programa.
162
163       SELECT SINGLE pgmna
164         INTO program
165         FROM tstc
166       WHERE tcode = t_code.
167
168       PERFORM read_prog.
169     ENDLOOP.
170
171   END-OF-SELECTION.
172
```

Em cada interação, recolhemos um nome de um programa e, recorrendo à sub-rotina "*read_prog*", por nós implementada e que passaremos a detalhá-la com mais pormenor no subcapítulo – *Parsing do código-fonte*, para ler e guardar todo o código ABAP desse mesmo programa numa estrutura interna "*gt_code*" criada para o efeito.

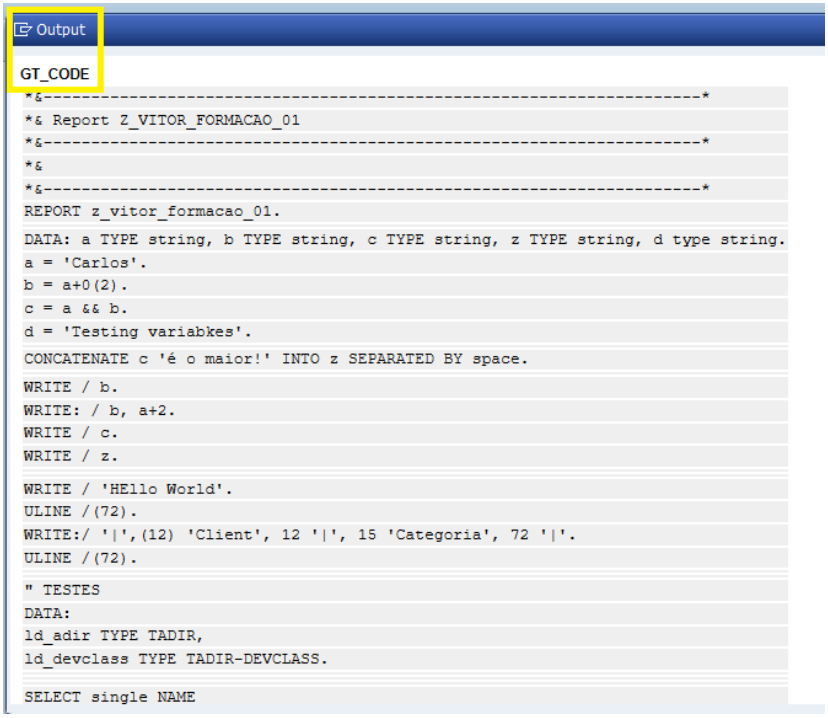


Figura 17 Conteúdo da estrutura gt_code

De seguida invocamos outra sub-rotina "*fill_result*", que é onde tratamos o código recolhido dos programas de maneira a obter os resultados desejados, neste caso todas as funções de um programa. Para isso recorremos às palavras chave do ABAP "*call function*", para localizarmos as funções invocadas por um programa.

```
237       procura = 'CALL FUNCTION'.
238     ENDIF.
239
240     FIND ALL OCCURRENCES OF REGEX procura
241       IN TABLE pt_code
242       RESPECTING CASE
243       RESULTS results.
244
```

Figura 18 Bloco de código para localizar palavras chave

Após termos obtido todas as ocorrências das palavras chave "*call function*" do código a analisar e retirado o nome das funções, fazemos uma consulta à tabela TFTIT onde a SAP guarda a informação sobre funções. E assim

conseguirmos obter uma descrição da função em causa e inserirmos essa mesma informação na estrutura interna "*wa_final*".

```
264       SELECT SINGLE stext
265         FROM tftit
266         INTO lv_desc
267        WHERE spras = 'EN'
268       AND funcname = lv_linel.
269
```

Figura 19 Bloco de código da query à atbela TFTIT

Parâmetros das funções

Utilizando outra vez um ciclo *Loop* e a mesma logica de pensamento, que temos vindo a seguir, vamos percorrer a tabela interna "*git_final*" e daí retirar todas as funções encontradas, para com uma *query* à tabela do SAP FUPARAREF obter a informação que desejamos sobre os parâmetros de cada função. Assim conseguimos, caracterizar as funções encontradas com seus parâmetros e tipos de parâmetros, entrada ou saída. Podíamos também apresentar o campo de tipo de dados, mas é um campo da tabela FUPARAREF que nem sempre contém dados.

Classes

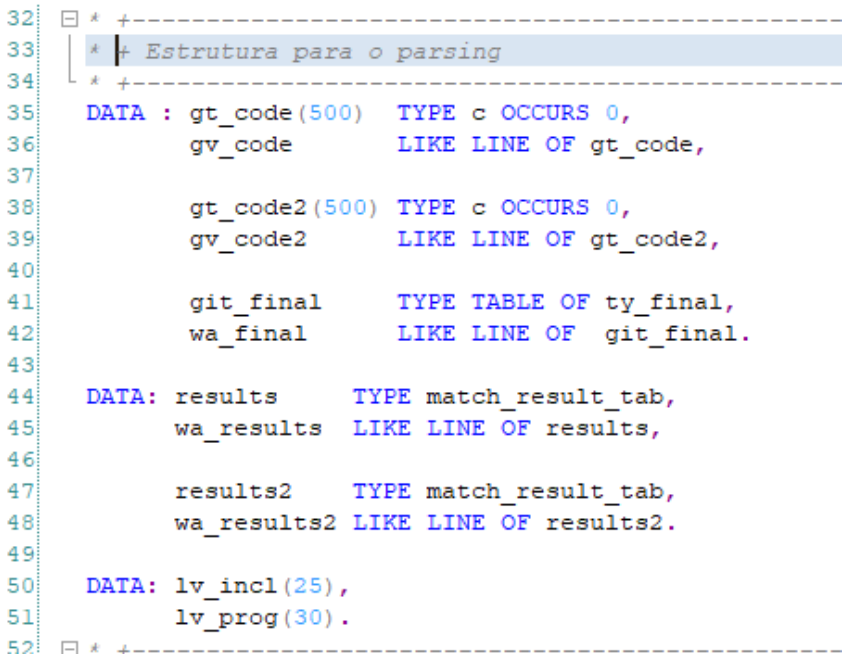
De forma a analisarmos as classes utilizadas no programa a ser analisado, recorremos à classe global pré-existente no SAP a "*cl_oo_class*", onde temos uma serie de método à nossa disposição de forma a obter a caracterização desejada das classes. Passamos a identificar os métodos que utilizamos:

GET_ATTRIBUTES	Devolve-nos os atributos da classe
GET_METHODS	Devolve-nos os componentes da classe
GET_COMPONENT_SIGNATURE	Devolve-nos os parâmetros e exceções dos componentes

GET_COMPONENT_EXPOSURE	Devolve-nos a visibilidade dos componentes
GET_ATTRIBUTE_TYPE	Devolve-nos os detalhes do tipo dos atributos
GET_SUBCLASSES	Devolve-nos os as subclasses
GET_SUPERCLASS	Devolve-nos as superclasses

Parsing do código-fonte

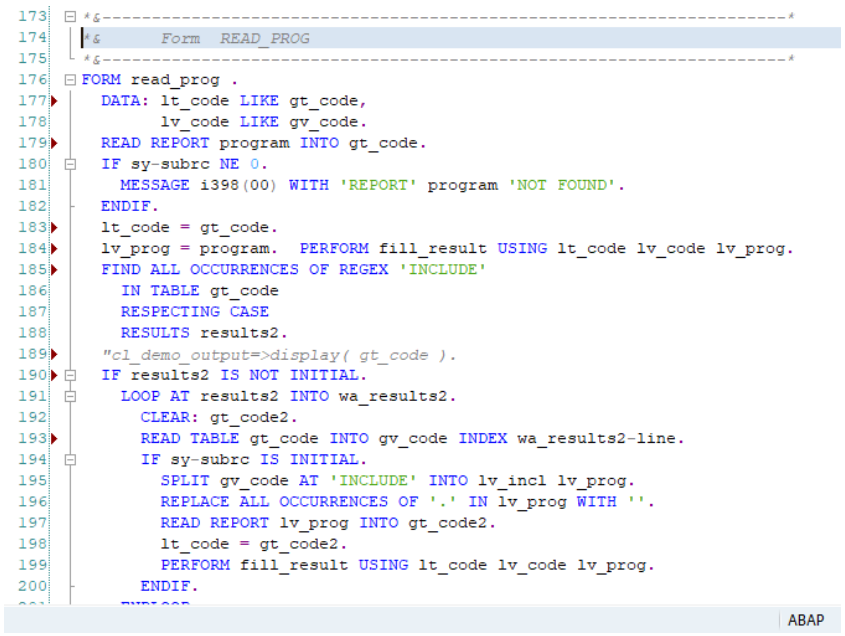
De forma a retirar alguma da informação, que pretendíamos optamos por realizar uma análise sintática, ou *parsing* em inglês, ao código-fonte do programa em análise e assim identificar palavras chave do ABAP, que nos permitam a partir daí obter a informação desejada. É também aqui, que reside no nosso entender algum trabalho futuro, pois se for desejado complementar a documentação gerada bastará adicionarmos ou criarmos por exemplo um dicionário, que identifique outras palavras-chave e traduza-as da forma mais conveniente para o programador e para a informação, que este deseje na documentação do software. Seria um trabalho interessante a fazer em conjunto com programadores SAP / ABAP.



```
* +-------------------------------------------------
* + Estrutura para o parsing
* +-------------------------------------------------
DATA : gt_code(500)  TYPE c OCCURS 0,
       gv_code       LIKE LINE OF gt_code,

       gt_code2(500) TYPE c OCCURS 0,
       gv_code2      LIKE LINE OF gt_code2,

       git_final     TYPE TABLE OF ty_final,
       wa_final      LIKE LINE OF  git_final.

DATA: results     TYPE match_result_tab,
      wa_results  LIKE LINE OF results,

      results2    TYPE match_result_tab,
      wa_results2 LIKE LINE OF results2.

DATA: lv_incl(25),
      lv_prog(30).
* +-------------------------------------------------
```

Figura 20 Bloco de código definição de estruturas

Começamos por criar uma estrutura interna capaz de guardar toda a informação ou todo o código-fonte de um determinado programa, e separamos o processo em duas fases ou sub-rotinas por nós implementadas "*Read_prog*" e "*Fill_result*".



```
*&---------------------------------------------------------------------*
*&      Form  READ_PROG
*&---------------------------------------------------------------------*
FORM read_prog .
  DATA: lt_code LIKE gt_code,
        lv_code LIKE gv_code.
  READ REPORT program INTO gt_code.
  IF sy-subrc NE 0.
    MESSAGE i398(00) WITH 'REPORT' program 'NOT FOUND'.
  ENDIF.
  lt_code = gt_code.
  lv_prog = program.  PERFORM fill_result USING lt_code lv_code lv_prog.
  FIND ALL OCCURRENCES OF REGEX 'INCLUDE'
    IN TABLE gt_code
    RESPECTING CASE
    RESULTS results2.
  "cl_demo_output=>display( gt_code ).
  IF results2 IS NOT INITIAL.
    LOOP AT results2 INTO wa_results2.
      CLEAR: gt_code2.
      READ TABLE gt_code INTO gv_code INDEX wa_results2-line.
      IF sy-subrc IS INITIAL.
        SPLIT gv_code AT 'INCLUDE' INTO lv_incl lv_prog.
        REPLACE ALL OCCURRENCES OF '.' IN lv_prog WITH ''.
        READ REPORT lv_prog INTO gt_code2.
        lt_code = gt_code2.
        PERFORM fill_result USING lt_code lv_code lv_prog.
      ENDIF.
```

ABAP

Figura 21 Bloco de código Sub-rotina Read_prog

Com a primeira sub-rotina vamos recolher todo o código-fonte de um dado programa e depois do mesmo ser guardado na tabela interna para o efeito

executamos a segunda rotina que irá propriamente dito realizar a analise sintática. Salvaguardamos aqui nesta sub-rotina a hipótese de existir algum código-fonte suplementar com a alguma inclusão de outro código-fonte e se assim for ou melhor se for detetada a palavra-chave *include* é retirado o nome do programa onde irá estar o outro código-fonte a ser guardado na tabela interna para posterior analise sintática na sub-rotina "*fill_result*".

Na Sub-rotina "*fill_result*", porque iremos fazer uma análise linha a linha do código-fonte, procurar palavras-chave, verificar tamanho de *strings* e de *chars* foi necessário criarmos variáveis para nos dar suporte nestas tarefas que irão permitirmos fazer a analise sintática da forma como projetamos. Começando a sub-rotina "*fill_result*" utilizando as tabelas pré preenchidas pela sub-rotina "*Read_prog*", ou seja, já com o código-fonte do programa a ser analisado passamos à procura das palavras-chave que referenciem os objetos que necessitamos para gerar conteúdo desejado na documentação do software a ser gerada.

```
221  FORM fill_result  USING  pt_code LIKE gt_code
222                           pv_code LIKE gv_code
223                           pv_prog LIKE lv_prog.
224    DATA: lv_line(100),
225          lv_line1(100),
226          lv_waste(50),
227          lv_i          TYPE i,
228          lv_idx        TYPE i,
229          lv_desc       TYPE rs38l_ftxt,
230          procura TYPE string.
231
232    IF check_class = abap_true.
233      procura = 'TYPE REF TO'.
234    ELSE.
235      procura = 'CALL FUNCTION'.
236    ENDIF.
237    FIND ALL OCCURRENCES OF REGEX procura
238      IN TABLE pt_code
239      RESPECTING CASE
240      RESULTS results.
241    LOOP AT results INTO wa_results.
242      CLEAR: lv_line1, lv_idx.
243
244      READ TABLE pt_code INTO pv_code INDEX wa_results-line.
245      SPLIT pv_code AT procura INTO pv_code lv_line.
246      lv_i = strlen( lv_line ).
247      lv_i = lv_i - 1.
```

Figura 22 Bloco de código Sub-rotina Fill_result

Depois de termos as referências que identificam a informação, que pretendemos no caso atual da nossa solução são, os nomes das funções e das

classes, guardamos em tabelas internas criadas para o efeito para posteriormente percorrermos essas mesmas tabelas e caracterizarmos individualmente todas as encontradas no código-fonte.

Output/Display

Optamos por apresentar o documento final em formato Word pois esta é uma das ferramentas de produtividade mais comum nos dias de hoje. E também por ser uma forma de a nossa solução se distinguir de todas as outras pois não encontramos nenhuma com esta opção.

Para o efeito recorremos ao OLE, que possibilita uma aplicação manipular objetos implementados noutra aplicação ou exponha-os para, que estes possam ser manipulados. A exposição permite que os clientes de automação automatizem determinadas funções, acedendo diretamente aos objetos e utilizar os serviços que estes fornecem. O objeto a ser automatizado pode ser local ou remoto. O OLE funciona com objetos incorporados num documento composto no qual apenas uma única parte do conteúdo pode estar ativa. Com a contenção ativa do documento, ativamos um documento inteiro dentro do contexto de um único frame. As extensões são interfaces adicionais, que permitem que um objeto *in-place* incorporável represente um documento inteiro em vez de um único conteúdo incorporado. A contenção ativa de documentos usa um *container* que fornece o espaço de exibição para documentos ativos e servidores que fornecem a interface do utilizador e os recursos de manipulação dos próprios documentos ativos.

O ABAP, suporta a técnica de automação OLE2 com base na interface de objeto aberto. Aplicativos *Desktop,* que disponibilizam as suas funções como um servidor de automação OLE2 como o Microsoft Excel e Microsoft Word podem, portanto, ser integrados ao sistema ABAP. No caso de uma invocação a partir de um programa ABAP, o SAP GUI atua como o cliente OLE e o aplicativo da área de trabalho como o servidor OLE.

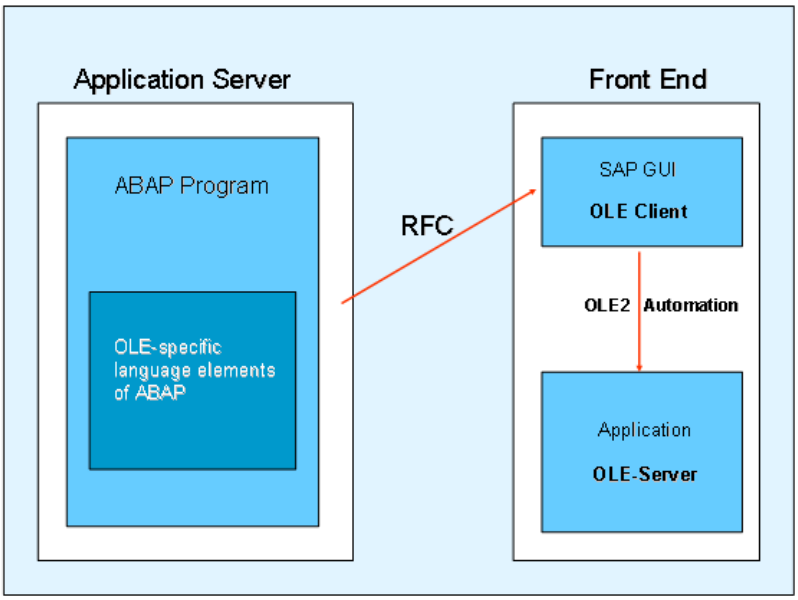


Figura 23 OLE quando invocado de um programa ABAP, o SAP GUI atua como cliente OLE e o aplicativo da área de trabalho como o servidor OLE

Para nos permitir manipular todas partes de um documento ativo, declaramos variáveis do tipo objetos OLE2, para assim moldar as diferentes partes do documento e nos permitir apresentar a documentação no formato por nós idealizado.

```
96    *& Doc Word Session
97    *&---------------------------------------
98    DATA: v_objword      TYPE ole2_object, "
99          v_documents    TYPE ole2_object, "
100         v_document     TYPE ole2_object, "
101         v_objselection TYPE ole2_object, "
102         v_tables       TYPE ole2_object, "
103         v_range        TYPE ole2_object, "
104         v_table        TYPE ole2_object, "
105         v_table_border TYPE ole2_object, "
106         v_table_cell   TYPE ole2_object, "
107         v_font         TYPE ole2_object ,
108         v_paragraph    TYPE ole2_object, "
109         v_parformat    TYPE ole2_object ,
110         v_pos(5)       TYPE n . "Position
111
112   DATA : tb_lines   TYPE i,
113          tb_counter TYPE i.
114
115   DATA funcoes LIKE LINE OF git_final.
```

Figura 24 Bloco de código declaração dos objetos OLE2

No momento só o modelo de objeto OLE2, aquele utilizado por nós, é suportado daí termos declarado todos os objetos tipo OLE2_OBJECT.

As seguintes palavras-chave ABAP, permitem processar objetos externos com o ABAP:

- CREATE OBJECT
- SET PROPERTY
- GET PROPERTY
- CALL METHOD
- FREE OBJECT

Com a palavra-chave CREATE OBJECT, o processador ABAP lê a entrada relevante na tabela TOLE e usa o *id* de classe OLE encontrado para chamar a função *ole_create_object* no *frontend* via RFC. Em seguida as funções, que são necessárias para gerar um objeto OLE são chamadas da biblioteca OLE do Windows e um identificador de objeto é retornado para o processador ABAP. Os comandos *call method, get property e set property* são agrupados no servidor de aplicações e passados para a função da *sapgui ole_flush_call* na forma de uma tabela interna. A palavra chave GET PROPERTY, copia a propriedade "X" do objeto "XY" para o campo "Y". E a palavra chave SET PROPERTY, define a propriedade "X" do objeto "XY" de acordo com o conteúdo do campo "Y".

```
339     GET PROPERTY OF v_objword 'Selection' = v_objselection.
340     GET PROPERTY OF v_objselection 'ParagraphFormat' = v_parformat.
341     SET PROPERTY OF v_parformat 'Style' = -2.
342     CALL METHOD OF
343       v_objselection
344       'TypeText'
345       EXPORTING
346         #1 = ' Environment Analysis of    _'.
```

Figura 25 Bloco de código exemplo da utilização das palavras chave OLE2

Documento gerado/output

Como resultado ou output do gerador de documentação, Environment Analysis, obtemos um documento Word estruturado da forma acima descrito, onde são aplicados estilos de cabeçalho, aos cabeçalhos no corpo do documento

gerado, permitindo, assim, recorrendo ao painel de navegação uma leitura, consulta, dinâmica e objetiva. A navegação no documento torna-se prática, e simples mesmo em documentos extensos, e por ter um formato Word (doc) permite também a edição do documento por parte do utilizador, dando a oportunidade de adicionar conteúdos, que este queira ver ou atualizar, mantendo inalterada a informação gerada. Assim começa com a hierarquia de programas, ou seja, lista todos os programas e inclusões existentes no programa, e são apresentadas como sub-nós do programa analisado. Apresenta o nome, título, descrição e lista de eventos definidos no programa. Dando ainda a oportunidade ao utilizador de analisar um pacote (com vários programas) ou apenas um programa individual, uma das opções terá sempre de ser escolhida.

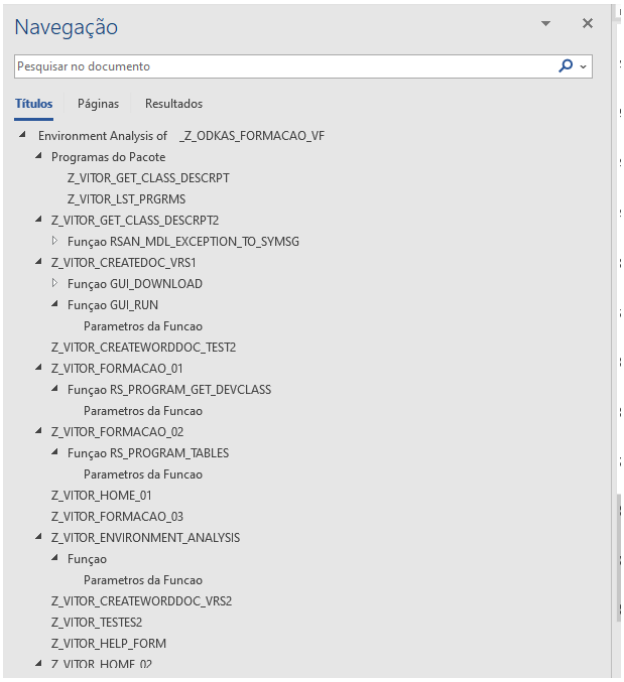


Figura 26 Painel de Navegação de um documento gerado pelo Environmet Analysis – Hierarquia de programas

Seguem-se as tabelas utilizadas no programa com a seguinte informação, nome e título da tabela e tipo de acesso. No que tocas às classes apresenta o nome a sua declaração e descrição, atributos, métodos, exceções, detalhes das sub classes e lista as super classes, tudo baseado na definição da classe.

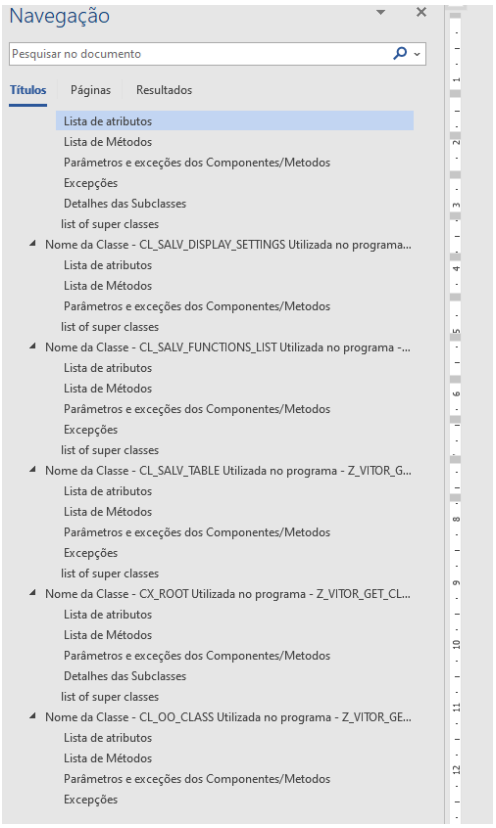


Figura 27 Painel de Navegação de um documento gerado pelo Environmet Analysis – Hierarquia de classes

O programador/utilizador, para atingir estes resultados tem a oportunidade no menu inicial de optar por selecionar qual a informação a ver expressa no documento gerado, é lhe dada também a opção de configurar o output ou a apresentação da escrita no documento, ou seja, em formato escrita em parágrafos ou em tabelas.

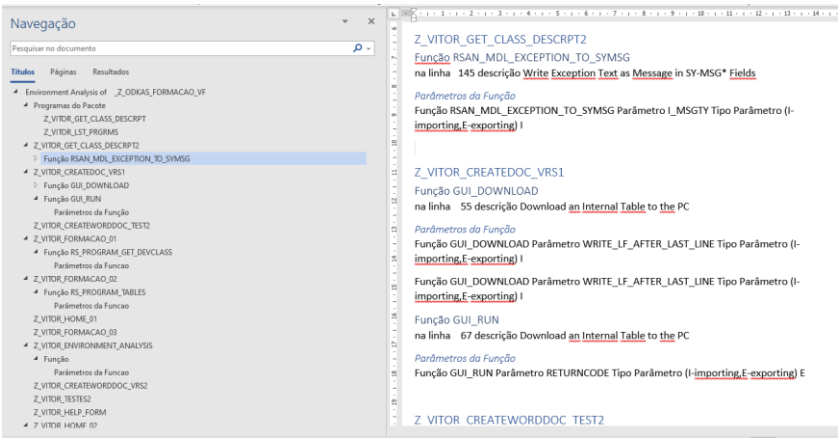


Figura 28 Escrita em parágrafos

Lista de atributos

Nome da Classe	Componente/Método
CL_OO_OBJECT	ATTRIBUTES
CL_OO_OBJECT	CLSKEY
CL_OO_OBJECT	COMPONENTS
CL_OO_OBJECT	EVENTS
CL_OO_OBJECT	EVENT_PARAMETERS
CL_OO_OBJECT	METHODS
CL_OO_OBJECT	METHOD_EXCEPTIONS
CL_OO_OBJECT	METHOD_PARAMETERS
CL_OO_OBJECT	SUBCOMPONENTS

Lista de Métodos

Classe	Métodos
CL_OO_OBJECT	GET_ATTRIBUTES
CL_OO_OBJECT	GET_ATTRIBUTE_TYPE
CL_OO_OBJECT	GET_COMPONENT_EXPOSURE
CL_OO_OBJECT	GET_COMPONENT_SIGNATURE
CL_OO_OBJECT	GET_EVENTS
CL_OO_OBJECT	GET_INSTANCE
CL_OO_OBJECT	GET_METHODS
CL_OO_OBJECT	GET_PARAMETER_TYPE
CL_OO_OBJECT	IS_COMPONENT_STATIC
CL_OO_OBJECT	IS_METHOD_ABSTRACT
CL_OO_OBJECT	IS_METHOD_EVENTHANDLER
CL_OO_OBJECT	IS_METHOD_FINAL

Parâmetros e exceções dos Componentes/Metodos

Figura 29 Escrita em tabelas

Capítulo 5
Avaliação

Das fases mais importantes num projeto de investigação é a validação dos conceitos teóricos desenvolvidos. Pois, é importante estabelecer um paralelismo numa determinada situação entre a abordagem proposta e a pré-existente. Este capítulo procura explicar o contexto, e procedimentos efetuados para levar a cabo a avaliação, e posterior validação do trabalho desenvolvido ao mesmo tempo que identifica os principais resultados obtidos.

Apesar da documentação de um código-fonte de qualquer programa ser da máxima importância, os programadores não as utilizam frequentemente, por variadas razões muitas já detalhadas ao longo deste documento, talvez por isso não existam métricas específicas para a avaliação da qualidade das ferramentas de geração de documentação.

Haveria diversas abordagens possíveis, para realizarmos este processo de avaliação, optamos por nos focar essencialmente na avaliação da usabilidade tendo em mente a norma ISO 9241 – 11 onde redefine a definição de usabilidade como "…*a capacidade de um produto ser utilizado por utilizadores específicos para atingir objetivos específicos com eficácia, eficiência e satisfação dentro de um contexto específico de utilização*.". A norma também esclarece os conceitos de;

- Utilizador – *pessoa que interage com o produto.*
- Contexto de utilização – *utilizadores, tarefas, equipamentos, ambiente físico e social em que o produto é utilizado.*
- Eficácia – *precisão e completude com que os utilizadores atingem objetivos específicos, acedendo a informação correta ou criando os resultados esperados.*
- Eficiência - *precisão e completude com que os utilizadores atingem seus objetivos em relação à quantidade de recursos gastos.*
- Satisfação – *conforto e aceitação do produto, medido por meios de métodos subjetivos e/ou objetivos.*

Iremos introduzir, e descrever a metodologia por nós seguida, ao longo deste capítulo, que combina vários métodos de IHC (Interação humano-computador)

de forma a tirar o máximo de proveito dos pontos fortes de cada método, e assim com esta combinação conseguirmos adquirir uma visão alargada da usabilidade da solução proposta por nós neste trabalho. A avaliação decorrerá em três fases; um workshop, uma avaliação heurística e uma análise subjetiva.

Metodologia

A metodologia seguida para avaliar a usabilidade do gerador de documentação por nós apresentado, combina vários métodos de avaliação de usabilidade; observação, avaliação heurística, questionário e entrevistas. A combinação destes métodos permitir-nos-á ter uma visão alargada da usabilidade do *Environment Analysis.*

De maneira a completar os resultados recolhidos, foram também recolhidos como métricas de análise o tempo individual e total necessário para realizar as tarefas pretendidas, como também a consistência de informação entre os conteúdos. Outras métricas utilizadas foram a qualidade da documentação produzida e o grau de definição dos processos.

Os documentos utilizados para validar esta abordagem fazem parte da documentação técnica deste projeto.

Papéis

Na nossa abordagem metodológica identificamos três papéis fundamentais;

- ***Programadores Especialistas***, engenheiros informáticos com experiência e conhecimentos em ABAP e SAP. Participam no workshop (1ª fase do processo de avaliação), são utilizados na avaliação heurística e na análise subjetiva.
- ***Programadores Principiantes***, engenheiros informáticos com experiência em programação, *com algum conhecimento sobre a linguagem ABAP*. Participam no workshop (2ª fase do processo de avaliação), são utilizados na avaliação heurística e na análise subjetiva.

- ***Avaliadores,*** engenheiros informáticos que recolhem e analisam resultados obtidos em cada fase da avaliação. Participam ativamente no workshop (3ª fase do processo de avaliação) e conduzem entrevistas.

Visão geral

Elaboramos um gráfico representativo da nossa abordagem ao processo de avaliação, expresso na figura a seguir, onde podemos ver todos os passos dados ao longo desse processo.

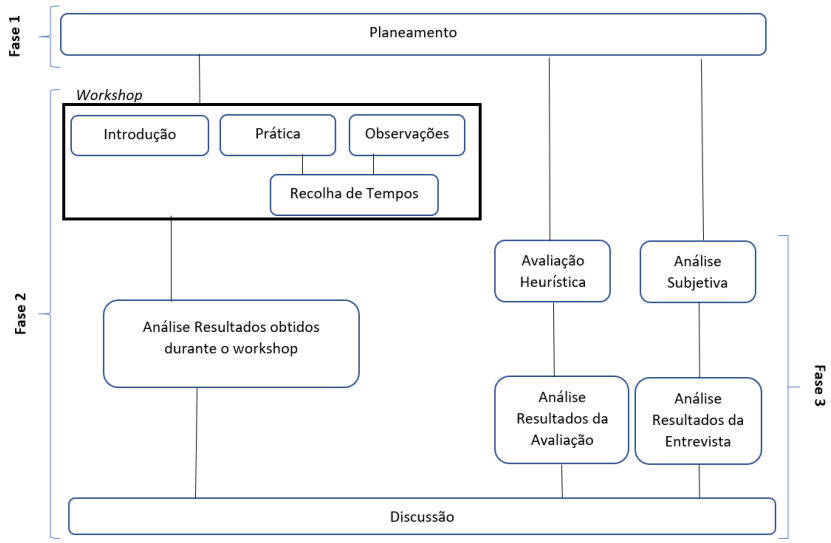


Figura 30 Metodologia

Fase 1: Planeamento, nesta fase o avaliador recruta alguns programadores, especialistas e principiantes, que irão estar envolvidos ao longo do processo de avaliação. São fixados os objetivos a alcançar e realçados os aspetos da solução a avaliar, como também os detalhes das fases seguintes.

Fase 2: Workshop, aqui convidamos todos os intervenientes do processo de avaliação a trabalharem com a nossa solução, *Environment Analysis,* dividimos a ordem de trabalhos em duas partes; uma primeira consistindo numa introdução sobre o tema da geração de documentação e o *Environment Analysis*; uma segunda parte onde convidamos os programadores principiantes e especialistas a gerarem documentação recorrendo ao nosso gerador de documentação é lhes pedido também que vão, tomando notas sobre as tarefas realizadas e dificuldades que possam surgir ao realiza-las. Os avaliadores apesar de não participarem

ativamente no workshop, estiveram presentes tirando notas sobre problemas de usabilidade, cronometrando algumas das tarefas e apoiando todos os que se depararam com dificuldades.

Fase 3: Avaliação heurística e análise subjetiva, após o workshop é realizada uma avaliação heurística e de seguida a analise subjetiva composta por questões e recolha de opiniões para análise.

Por fim é realizada a análise final e discussão dos resultados obtidos neste processo. O resultado desta metodologia é um conjunto de recomendações para melhoramento da ferramenta e validação da mesma como um instrumento válido na geração de documentação.

Estudo de caso

Estudo de caso é uma das várias estratégias de investigação onde um caso pode ser algo bem definido ou concreto, mas também pode ser algo menos definido ou definido num plano mais abstrato como, decisões, programas, processos de implementação ou mudanças organizacionais, Rodríguez (1999) [13], Yin (1993 e 2005) [14] e Stake (1994) [15]. A vantagem do estudo de caso é a sua aplicabilidade a situações humanas, a contextos contemporâneos de vida real (Dooley, 2002) [16]. Não é uma tática de recolha de dados, nem uma característica de planeamento de pesquisa, é um método abrangente e alcança o processo completo, como, o planeamento, abordagens à recolha e analise dos dados.

O estudo de caso é definido como, "*uma investigação empírica que investiga um fenômeno contemporâneo (o 'caso') em profundidade e dentro de seu contexto do mundo real*" Yin (2015) [17].

O estudo de caso só faz sentido se estiver assente num rigoroso desenho metodológico, partindo de um problema iniciado com "porquê" ou "como" onde objetivos e enquadramento teórico da investigação terão de ser claros. O problema em questão poderá dividir-se em proposições que por sua vez se dividirão em questões orientadoras. A identificação das unidades de análise é obrigatória e o desenho dos instrumentos de recolha da informação. Deverá

também realizar-se o necessário registo e classificação da informação a partir das múltiplas fontes de evidência; realizar a triangulação da informação para dar resposta às questões orientadoras e, por último, filtrar criticamente a problemática estudada com os elementos conceptuais teóricos que fundamentaram o estudo.

O estudo de caso, oferece-nos uma abordagem útil e pertinente para combinarmos várias fontes de evidências, como observação, documentação e entrevistas, segundo Hamel, *"O estudo de caso faz recurso a uma diversidade de formas de recolha de informação, dependente da natureza do caso e tendo por finalidade, possibilitar o cruzamento de ângulos de estudo ou de análise"* (Hamel, 1997) [18], exatamente aquilo por nós pretendido nesta avaliação.

Descrição Estudo

Seguindo a abordagem metodológica por nós descrita na figura nº 30, preparamos um estudo de caso múltiplo, permitindo assim a obtenção de resultados de diferentes métodos de avaliação. Realizamos um workshop com o objetivo de aprofundarmos a usabilidade da nossa solução. A avaliação heurística para recolhermos descobertas identificadas pelos programadores especialistas e principiantes, adicionalmente também os entrevistamos individualmente com o objetivo de recolher dados qualitativos sobre a usabilidade do *Environment Analysis*. Ambos os dados qualitativos e quantitativos foram recolhidos para analise e como fonte para suporte de potenciais problemas de usabilidade e por conseguinte também a validação da nossa solução como uma ferramenta sólida.

Iniciamos o workshop com um, esclarecimento sobre os requisitos e objetivos do mesmo, como também, informação detalhada sobre o funcionamento da solução proposta neste trabalho. Foi facultado aos programadores principiantes e especialistas todos os recursos necessários para a realização do workshop juntamente com um questionário demográfico focado na experiência em programação.

Foram recrutados seis programadores cinco sem e um com experiência em SAP e ABAP, foi-lhes facultado um acesso ao *Environment Analysis*, após feita uma pequena abordagem sobre a ferramenta é lhes dado uma lista de tarefas a realizar.

Foram consideradas três tarefas:

- Gerar documentação referente a analise de um pacote ou um programa com um scope mais reduzido e predefinido no menu inicial;
- Gerar documentação referente a analise de um pacote (apenas programas);
- Gerar documentação referente a analise de um programa à escolha, mas com um scope alargado (4 opções mínimo);

Juntamente com as tarefas de cenário propostos foi entregue um questionário elaborado tendo por base o questionário desenvolvido por Lewis [19].

A avaliação heurística teve por base os dez princípios de Jacob Nielsen [20] para o design de interação com o utilizador. Na avaliação realizada, logo após o workshop, e com o objetivo de identificar o que melhorar, manter e complementar na nossa interface, foi facultado aos programadores um formulário com uma nota introdutória às dez heurísticas de usabilidade e o significado destas, uma tabela com os níveis de severidade de forma a classificar cada heurística e uma tabela com cada heurística.

Tabela 1 Níveis de Severidade

Nível de severidade	**Tipo**	**Descrição**
0	Sem importância	Não é um problema de usabilidade impactante.
1	Cosmético	Apenas um problema cosmético sem grande impacto.
2	Simples	Pequeno problema de usabilidade, pode ser corrigido.
3	Grave	Grande problema de usabilidade, deve ser corrigido.
4	Catastrófico	Problema de usabilidade catastrófico, necessária correção urgente.

Tabela 2 Avaliação Heurísticas

	Heurística	Descrição	Grau atribuído
H1	DIÁLOGO SIMPLES E NATURAL	O sistema exibe o necessário para o utilizador? Há informação em excesso apresentada?	
H2	FALAR A LINGUAGEM DO UTILIZADOR	O sistema utiliza uma linguagem familiar ao utilizador?	
H3	MINIMIZAR A CARGA DE MEMÓRIA DO UTILIZADOR	O utilizador precisa lembrar-se de informações entre as funcionalidades do sistema?	
H4	CONSISTÊNCIA	As ações têm os mesmos resultados em diferentes situações?	
H5	FEEDBACK	Os utilizadores são informados sobre o status das solicitações em tempo útil?	
H6	SAÍDAS EVIDENTES	O sistema permite aos utilizadors meios para cancelar ações que não são mais desejadas?	
H7	ATALHOS	O sistema oferece meios para executar ações de forma otimizada?	
H8	MENSAGENS DE ERRO	O sistema oferece mensagens de erro claras e de fácil compreensão?	
H9	PREVENÇÃO DE ERROS	O sistema possui falhas de projeto, erros que poderiam ser facilmente evitados?	
H10	DOCUMENTAÇÃO E AJUDA	O sistema oferece informações para ajuda claras, precisas e de fácil localização?	

No final da avaliação, os entrevistados responderam ainda a um pequeno questionário de satisfação.

A avaliação subjetiva foi conduzida pelo avaliador após a avaliação heurística, individualmente com cada programador, tendo por base um questionário e suas respostas. Ao longo da entrevista, foram focados vários aspetos de usabilidade do sistema, tais como, a sua compreensão, a sua operacionalidade, eficiência, robustez e satisfação. Foram tiradas notas ao longo da entrevista pelo avaliador e recolhidas sugestões deixadas pelos programadores.

Resultados e Análise

Nesta secção vamos apresentar e analisar os resultados obtidos, durante o processo de avaliação da solução aqui apresentada neste trabalho. De referir que o universo dos participantes no processo foi composto por 83% com grau académico de licenciado, e 17% com o grau de mestrado. Com idades, entre os, 25 aos 34 anos - 67%, 35 aos 44 - 17%, dos 18 aos 24 - 16%, e 100% do género

masculino. A experiência em programação, variou entre os 5 e os 10 anos conforme podemos observar no gráfico da figura 32.

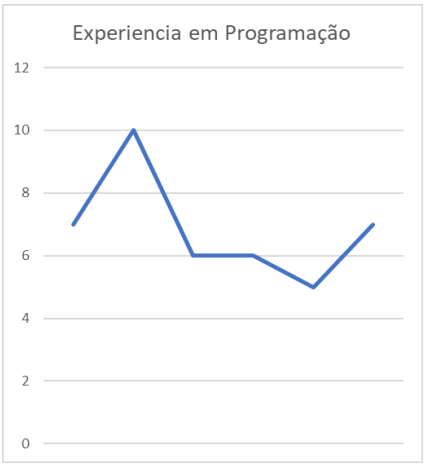


Figura 31 Experiência anos em programação

Workshop; Durante o workshop foi sendo recolhido diversos dados através de apontamentos de notas, aquando a realização das tarefas propostas e das respostas ao questionário.

Foi pedido a todos, que preenchessem o questionário *after-scenario questionnaire,* a fim de apurar três aspetos importantes em relação à satisfação do utilizador: dificuldade em realizar as tarefas; duração a concluir as tarefas; informação de suporte para concluir as tarefas. Estes aspetos foram medidos numa escala *Likert Scale* de 5 pontos, sendo o extremo esquerdo de "Discordo Plenamente" e o extremo direito "Concordo Plenamente", os dados foram tratados e o resultado expresso na figura 33. Os resultados mostram-nos, que apesar de o programador experiente não ter mostrado nenhuma dificuldade em realizar as tarefas no que toca aos principiantes, apesar de, terem expresso não ter dificuldades em concluir a maior parte das tarefas, a tarefa 3 foi a de maior dificuldade em concluir.

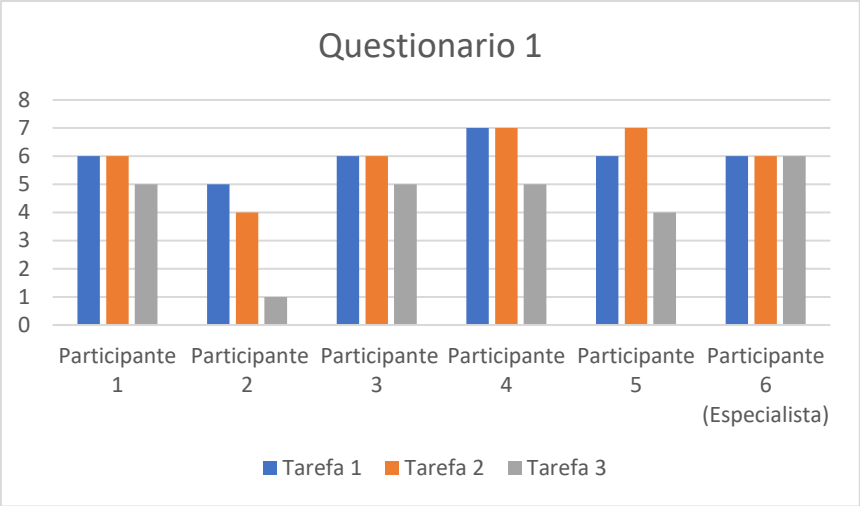


Figura 32 Resultados do After-Scenario Questionnaire

As questões levantadas na realização das tarefas foram, principalmente relacionadas com questões de documentação e interpretação de algumas opções, por parte dos programadores principiante. Ao programador especialista não foi necessário, mas ao longo do workshop o avaliador foi dando apoio com explicações necessárias e exemplificações.

Avaliação heurística; foi pedido a todos os participantes, que respondessem ao questionário de avaliação das heurísticas, os resultados obtidos foram recolhidos e analisados pelo avaliador. Os problemas encontrados nos resultados obtidos na avaliação das dez heurísticas, foram maioritariamente com as heurísticas H2, H6, H9 e H10, que representam; Falar a linguagem do utilizador, o menu de opções não foi percetível a todos; Saídas evidentes, falta de um botão para cancelar a geração do documento; Prevenção de erros, não previne todos os erros; Documentação e ajuda, necessidade de mais documentação.

	P1	P2	P3	P4	P5	P6
H1	0	0	0	1	2	0
H2	0	3	3	2	1	0
H3	2	0	1	0	1	0
H4	1	0	2	0	1	0
H5	1	0	1	0	2	2
H6	0	4	0	3	2	3
H7	1	1	0	0	1	0
H8	0	4	2	0	2	0
H9	1	2	3	2	0	2
H10	0	1	1	4	1	0

Figura 33 Resultados Avaliação Heuristica

Encontramos também, alguns problemas ao nível da H8, mensagens de erro, apesar do programador especialista não ter avaliado dessa forma.

No final da avaliação heurística existiu um pequeno questionário, de forma a conseguirmos obter feedback da interação com o gerador de documentação, em termos de manuseamento da ferramenta, a grande maioria dos participantes considerou fácil e a documentação gerada clara e concisa.

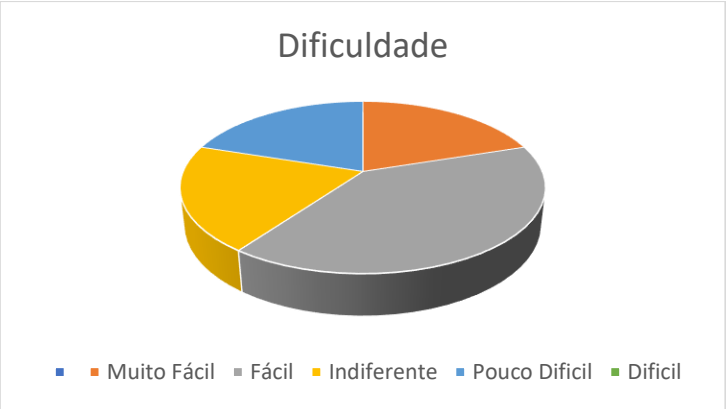


Figura 34 Resultado do questionário final das heurísticas

Avaliação subjetiva; as entrevistas permitiram-nos adquirir informação mais detalhada e de forma mais aprofundada, sobre a utilização da ferramenta de geração de documentação apresentada, trazendo também sugestões por parte dos participantes ao melhoramento da ferramenta.

Novamente, nesta avaliação os problemas já detetados na avaliação anterior das heurísticas, foram realçados, ainda assim a maioria considerou o sistema não complexo, fácil de usar, com funções bem integradas, bastante confiantes ao utilizar a ferramenta e sem a necessidade de adquirir bastante

conhecimento antes de conseguir trabalhar com o gerador de documentação. Por fim foi pedido que respondessem a um último questionário relacionado com a utilização do gerador de documentação *Environment Analysis* com os resultados expressos na figura 36.

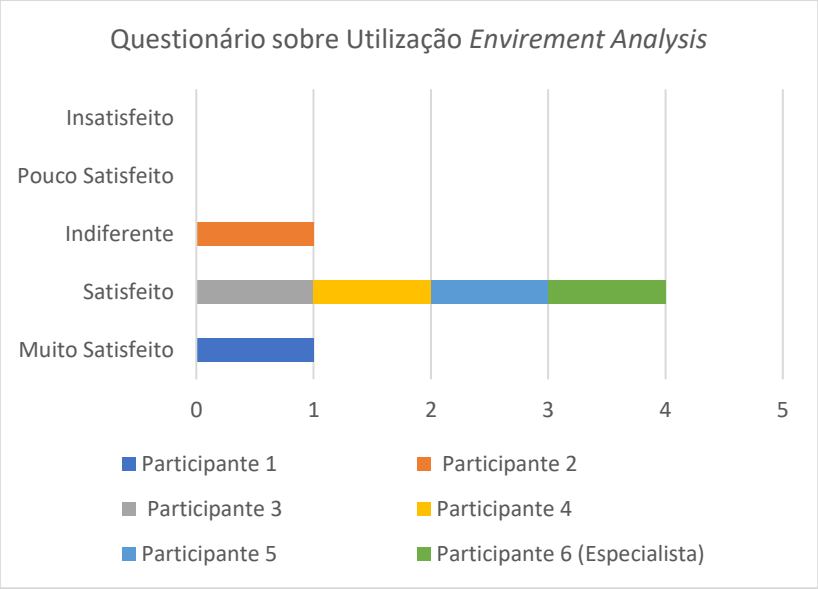


Figura 35 Resultados Questionário sobre utilização do Envirement Analysis

Discussão

Para nos ajudar a tirar as conclusões dos resultados, vamos contextualizar o estudo e passarmos a discussão das descobertas de cada parte da metodologia.

Cada método aqui aplicado tem seus pontos fortes e fracos e concentram-se em diferentes áreas de problemas. Neste tema da geração de documentação, a avaliação de uma ferramenta do género não se previa simples, pois para além da exploração do tema não ser frequente o "como" avaliar torna-se difícil. Ainda assim a combinação dos métodos selecionados resultou numa visão completa da usabilidade geral. A combinação dos métodos específicos permitiu, portanto, abordar problemas de usabilidade relacionados a uma ferramenta de geração de documentação de forma adequada.

Apesar da avaliação heurística envolver um pequeno grupo de especialistas em usabilidade, analisando uma ferramenta de geração de documentação e, examinando a sua conformidade com um conjunto predefinido

de heurísticas, na nossa metodologia a abordagem, programadores iniciantes e especialistas realizaram a avaliação heurística. Como afirma Molich e Jeffries em [21], a avaliação heurística pode ser aplicada por "*alguém sem conhecimento particular de engenharia de usabilidade*". Hertzum e Jacobsen também afirmaram em [22], que "*qualquer profissional de informática deve ser capaz de aplicar a avaliação heurística, mas a informalidade do método deixa muito para o avaliador*". De facto, os níveis de conhecimento e experiência de avaliadores individuais têm influências evidentes na avaliação da usabilidade resultados - o chamado efeito avaliador. Tentamos reduzir o efeito do avaliador, durante o workshop prático, fornecendo uma introdução aprofundada à documentação da ferramenta de geração de documentação e às suas partes funcionais (conceitos principais).

Além disso, na nossa abordagem metodológica, incluímos uma pequena amostra de participantes porque: o estudo de caso requer recursos e tempo extensos; e o nosso supervisor concordou que uma amostra pequena seria suficiente para tirar conclusões úteis e confiáveis.

Devido ao tamanho da amostra, as nossas conclusões têm uma generalização limitada.

Os resultados foram diferentes em cada parte da metodologia. Enquanto a avaliação heurística forneceu descrições formais e detalhadas dos problemas na ferramenta de geração de documentação, o workshop concentrou-se nos problemas de usabilidade, relacionados ao conceito e estrutura da ferramenta de geração de documentação.

Durante o workshop, pudemos revelar mais problemas de tempo de execução que não são óbvios e, portanto, não podem ser encontrados na avaliação heurística. As entrevistas não apenas forneceram uma compreensão mais profunda desses problemas, mas também revelaram suas opiniões subjetivas à ferramenta de geração de documentação.

Foram feitas algumas alterações, de forma a irmos de em encontro aos nossos resultados obtidos, a tabela 3 mostra-nos algumas dessas alterações.

Tabela 3 Alterações realizadas após avaliação

	Alterações
H1	
H2	Em vez de abreviaturas. Foi colocado os nomes por inteiro
H3	
H4	
H5	
H6	Botão de Cancelar e interromper a geração de documentação
H7	
H8	Foram acrescentadas novas mensagens de erro
H9	Foram acrescentadas mais validações
H10	Não alteramos por termos chegado a conclusão de que a causa era a não familiarização com o sistema SAP

Capítulo 6
Conclusão

Com a realização deste trabalho, pensamos ter contribuído para dar mais um passo no que diz respeito à produção eficaz de documentação com qualidade no âmbito de um projeto de software. Foram alcançados bons resultados no que diz respeito à usabilidade, disponibilidade, desempenho e segurança da aplicação e também à consistência da informação existente nos documentos gerados.

O conteúdo gerado no documento final, foi considerado por todos os participantes na avaliação como sendo conciso, considerando mesmo ser um documento que permite interpretar uma realidade bastante especifica, como sendo o SAP, e trazê-la para um domínio fora desse sistema tornando-se percetível a todos.

Em conclusão, podemos referir que este trabalho, conseguirá trazer valor para as organizações onde for aplicado. Destacando a sua abrangência mediante a conceptualização de uma solução genérica, a sua aplicação prática mediante a criação de uma ferramenta de suporte aplicável a um domínio específico e o facto de ter sido desenvolvido e testado com sucesso. Esta ferramenta ainda permitiu a obtenção de valores muito interessantes em termos de tempo e facilidade no processo de geração e manutenção de documentação com características visíveis de heterogeneidade.

Trabalho futuro

Mas todos os projetos têm espaço para melhoramentos e como tal este também, assim onde consideramos ser interessante desenvolver algum trabalho futuro seria:

- Encontrar formas de melhorar os ganhos de performance obtidos.
- Adicionar suporte para outros formatos de output

- Adicionar palavras-chave ao dicionário interno de forma a filtrar ainda mais o conteúdo encontrado no código e obter ainda mais detalhe no documento gerado
- Criar forma de documentar os DDIC (Data Element)

Referências bibliográficas

[1] 'Software Documentation - Building and Maintaining Artefacts of Communication'. [Online]. Available: http://www.site.uottawa.ca/~tcl/gradtheses/aforward/aforward_thesis.html. [Accessed: 23-Jan-2019]

[2] 'The Package Concept (SAP Library - ABAP Workbench Tools)'. [Online]. Available: https://help.sap.com/saphelp_nw70ehp1/helpdata/en/ea/c05da3f01011d3964000a0c94260a5/content.htm?no_cache=true. [Accessed: 17-Jan-2019]

[3] 'SAP Community Home', *SAP*. [Online]. Available: https://www.sap.com/community.html. [Accessed: 30-Jan-2019]

[4] Available: http://msoo.pbworks.com/f/Scott+W.+Ambler+-+Agile+Modeling.pdf. [Accessed: 09-Feb-2019]

[5] A. Jazzar and W. Scacchi, 'Understanding the Requirements for Information System Documentation: An Empirical Investigation', in *Proceedings of Conference on Organizational Computing Systems*, New York, NY, USA, 1995, pp. 268–279 [Online]. Available: http://doi.acm.org/10.1145/224019.224048

[6] A. Cockburn, *Agile Software Development*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[7] 'Literate Programming'. [Online]. Available: http://www.literateprogramming.com/. [Accessed: 09-Feb-2019]

[8] 'Working with Transaction Codes (SAP Library - Getting Started - Using SAP Software)'. [Online]. Available: https://help.sap.com/doc/saphelp_nw70ehp2/7.02.16/en-US/f9/e1a442dc030e31e10000000a1550b0/content.htm?no_cache=true. [Accessed: 12-Feb-2019]

[9] 'Strings - ABAP Keyword Documentation'. [Online]. Available: https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenstring.htm. [Accessed: 12-Feb-2019]

[10] A. de L. Cabral, F. Gyenge, and M. A. Bianchi, *Glossário de termos técnicos de*. Senac, 2018.

[11] Available: https://help.sap.com/saphelp_nw73EhP1/helpdata/en/ef/d94b78ebf811d295b100a0c94260a5/frameset.htm. [Accessed: 14-Feb-2019]

[12] 'The early years | SAP History | About SAP SE', *SAP*. [Online]. Available: https://www.sap.com/corporate/en/company/history/1972-1980.html. [Accessed: 14-Feb-2019]

[13] G. Rodríguez Gómez, E. García Jiménez, and J. Gil Flores, *Metodología de la investigación cualitativa*. Málaga: Aljibe, 2005.

[14] R. K. Yin, *Case study research: design and methods*, Fifth edition. Los Angeles: SAGE, 2014.

[15] R. E. Stake, 'Case Study: Composition and Performance', *Bull. Counc. Res. Music Educ.*, no. 122, pp. 31–44, 1994.

[16] L. M. Dooley, 'Case Study Research and Theory Building', *Adv. Dev. Hum. Resour.*, vol. 4, no. 3, pp. 335–354, Aug. 2002.
[17] R. K. Yin, *Estudo de caso*. Porto Alegre: Bookman, 2003.
[18] J. Hamel, *Etudes de cas et sciences sociales*. Montreal; Paris: L'harmattan, 1997.
[19] J. R. Lewis, 'IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use', *Int J Hum-Comput Interact*, vol. 7, no. 1, pp. 57–78, Jan. 1995.
[20] W. L. in R.-B. U. Experience, '10 Heuristics for User Interface Design: Article by Jakob Nielsen', *Nielsen Norman Group*. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/. [Accessed: 05-Dec-2019]
[21] '"Describing Usability Problems" Paper Summary - DialogDesign'. [Online]. Available: http://www.dialogdesign.dk/Summary2.htm. [Accessed: 06-Dec-2019]
[22] M. Hertzum and N. Jacobsen, 'The Evaluator Effect : A chilling fact about usability evaluation methods', *Int. J. Hum.-Comput. Interact.*, vol. 15, p. 183, Jan. 2003.

Anexos

Anexo 1

Termo de consentimento

O estudo em questão pretende recolher, de forma empírica, dados a respeito de um conjunto de heurísticas de acessibilidade proposto pelo autor do estudo Vitor Gavina Faria.

Todos os dados recolhidos serão utilizados única e exclusivamente para tal estudo, sendo confidenciais.

O meu nome e informações pessoais que possibilitam a minha identificação, como endereço e telefone, serão mantidos em sigilo.

O estudo será realizado em locais e horários a serem definidos de acordo com as minhas preferências e disponibilidades, tendo eu total liberdade de desistir a qualquer momento de participar do mesmo.

Finalmente, declaro, na presente data, ter idade maior ou igual a 18 anos e concordar com os termos dispostos nesse documento.

Responsável pelo estudo:
Vitor Gavina Faria
Aluno Mestrado Engenharia Informática nº 2110312

Nome do participante: _____

Assinatura do participante: _____

Local e data: _____, ____ de _____ de _____

Anexo 2

Género:

Masculino	
Feminino	
Outro	

Idade:

18 – 24	
25 - 34	
35 – 44	
45 – 54	
55 ou mais	

Grau Académico:

Licenciatura	
Mestrado	
Doutorado	

Experiência em Programação (Anos)

1 – 3	
4 – 6	
7 – 9	
9 ou mais	

Anexo 3

After-Scenario Questionnaire

Fonte: Lewis, J. R. (1995) *IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use.* **International Journal of Human-Computer Interaction**, 7:1, 57-78.

Programador Especialista ☐

Programador Principiante ☐

			1	2	3	4	5	6	7	
1.	Satisfeito com a facilidade em concluir as tarefas neste cenário	discordo fortemente								concordo plenamente
2.	Satisfeito com a duração de tempo levado a concluir as tarefas neste cenário	discordo fortemente								concordo plenamente
3.	Satisfeito com as informações de suporte (ajuda online, mensagens, documentação) ao concluir as tarefas	discordo fortemente	○	○	○	○	○	○	○	concordo plenamente

Anexo 4

Questionário Avaliação Heurísticas

DIALOGO SIMPLES E NATURAL

As interfaces devem ser simples, uma vez que a quantidade de recursos adicionais sobrecarrega e podem não ser necessárias naquele momento, deve corresponder a tarefa do utilizador de forma o mais natural possível, para facilitar a relação entre conceitos do utilizador e do computador, o ideal é mostrar exatamente a informação que o utilizador precisa.

FALAR A LINGUAGEM DO UTILIZADOR

A terminologia da interface do utilizador deve ser voltada para o utilizador e não para o sistema, portanto a linguagem utilizada deve ser o mais natural possível, evitando códigos do sistema. É necessário um cuidado na utilização de elementos não verbais, como ícones, para conservá-los intuitivos.

MINIMIZAR A CARGA DE MEMÓRIA DO UTILIZADOR

Para diminuir a necessidade de memorização do utilizador o sistema deve exibir elementos de diálogo para permitir que eles possam escolher a partir dos mesmos, assim como elementos de interface que possibilitem escolher ações.

CONSISTÊNCIA

O sistema deve garantir a integridade das suas funções, que deverão ter o mesmo efeito quando executadas, além de apresentar informações de acordo com um padrão estabelecido, facilitando assim o reconhecimento.

FEEDBACK

O sistema deve manter um diálogo contínuo com o utilizador, sobre o que o mesmo está fazendo e inserindo no mesmo, principalmente antes de ações definitivas como subscrição de arquivos ou exclusão de dados.

SAÍDAS EVIDENTES

O sistema deve sempre disponibilizar uma forma de saída de uma determinada ação para o utilizador, com isso a segurança do utilizador em explorar o sistema aumentará, facilitando o aprendizado de forma exploratória.

ATALHOS

Atalhos devem ser fornecidos para as ações frequentemente usadas, como abreviações, teclas de função ou comandos chaves, caixas de diálogos com teclas para acessar funções importantes.

MENSAGENS DE ERRO

As mensagens de erro devem ser claras e precisas, de forma que o utilizador entenda facilmente o ocorrido, auxiliando-o a resolver o problema se possível.

PREVENÇÃO DE ERROS

As situações propícias a erros devem ser previstas, como solicitações de dados que poderiam ser selecionados pelo utilizador, e evitadas.

DOCUMENTAÇÃO E AJUDA

É preferível que um sistema seja simples o suficiente para possibilitar sua utilização sem a necessidade de algum tipo de ajuda ou documentação, mas este objetivo normalmente não pode ser cumprido, portanto deve ter um texto de qualidade e informações bem estruturadas;

Aceda de forma exploratória o Envirement Analysis, verificando suas diversas funcionalidades, e com base nas regras heurísticas apresentadas realize a avaliação do ambiente. Atribua valores aos problemas encontrados de acordo com a tabela de níveis de severidade.

Nível de severidade	**Tipo**	**Descrição**
0	Sem importância	Não é um problema de usabilidade impactante.
1	Cosmético	Apenas um problema cosmético sem grande impacto.
2	Simples	Pequeno problema de usabilidade, pode ser corrigido.
3	Grave	Grande problema de usabilidade, deve ser corrigido.
4	Catastrófico	Problema de usabilidade catastrófico, necessária correção urgente.

Grau atribuído: 0 - Sem importância 1- Cosmético 2 – Simples 3 - Grave 4 - Catastrófico

Heurística	**Descrição**	**Grau atribuído**
DIÁLOGO SIMPLES E NATURAL	O sistema exibe o necessário para o utilizador? Há informação em excesso apresentada?	
FALAR A LINGUAGEM DO UTILIZADOR	O sistema utiliza uma linguagem familiar ao utilizador?	
MINIMIZAR A CARGA DE MEMÓRIA DO UTILIZADOR	O utilizador precisa lembrar-se de informações entre as funcionalidades do sistema?	
CONSISTÊNCIA	As ações têm os mesmos resultados em diferentes situações?	
FEEDBACK	Os utilizadores são informados sobre o status das solicitações em tempo útil?	
SAÍDAS EVIDENTES	O sistema permite aos utilizadors meios para cancelar ações que não são mais desejadas?	
ATALHOS	O sistema oferece meios para executar ações de forma otimizada?	
MENSAGENS DE ERRO	O sistema oferece mensagens de erro claras e de fácil compreensão?	
PREVENÇÃO DE ERROS	O sistema possui falhas de projeto, erros que poderiam ser facilmente evitados?	
DOCUMENTAÇÃO E AJUDA	O sistema oferece informações para ajuda claras, precisas e de fácil localização?	

Por favor, responda as questões abaixo:

1. Nível de satisfação observado ao utilizar o Envirement Analysis:

Muito satisfeito	
Satisfeito	
Indiferente	
Pouco satisfeito	
Insatisfeito	

2. Nível de dificuldade encontrado ao utilizar o Envirement Analysis:

Muito fácil	
Fácil	
Indiferente	
Pouco difícil	
Difícil	

Declaro que estou ciente que meus dados neste questionário serão utilizados com fins de avaliação da usabilidade do o Envirement Analysis

Assinatura/Rubrica do participante:

_____

Anexo 5

	Concordo fortemente				Discordo fortemente
1. Penso que irei utilizar o Sistema frequentemente	1	2	3	4	5
2. Sou da opinião que o Sistema é desnecessariamente complexo	1	2	3	4	5
3. Pensei que o Sistema era fácil de utilizar	1	2	3	4	5
4. Penso que irei necessitar de apoio técnico para conseguir utilizar o sistema	1	2	3	4	5
5. Sou da opinião que as várias funções do sistema estão bem integradas	1	2	3	4	5
6. Penso que existe muita inconsistência neste sistema	1	2	3	4	5
7. Imagino que a maioria das pessoas aprenderão a utilizar o sistema bastante rápido	1	2	3	4	5
8. Sou da opinião que o sistema é bastante pesado para utilizar	1	2	3	4	5
9. Senti-me bastante confiante ao utilizar o Sistema	1	2	3	4	5
10. Tive a necessidade de adquirir muito conhecimento, antes de conseguir trabalhar com o Sistema	1	2	3	4	5

Sugestões:

_____

_____

_____

_____

[i] Linguagem de programação principal para o desenvolvimento de programas para o SAP
[ii] Sistema integrado de gestão comercial transacional