UNIVERSIDADE DE ÉVORA

Mestrado em Engenharia Informática

Smart Module Builder
Automatic Creation of Modules for SugarCRM

João Carlos Selésio de Morais

Orientador: Luís Arriaga da Cunha

Évora, Fevereiro de 2013

UNIVERSIDADE DE ÉVORA

Mestrado em Engenharia Informática

Smart Module Builder

Automatic Creation of Modules for SugarCRM

João Carlos Selésio de Morais

Orientador: Luís Arriaga da Cunha

Évora, Fevereiro de 2013

AGRADECIMENTOS

A realização desta Dissertação de Mestrado só foi possível graças à colaboração e ao contributo, de forma directa ou indirecta, de várias pessoas, às quais gostaria de dedicar algumas palavras de agradecimento e profundo reconhecimento, em particular:

Ao Professor Luís Fernando Arriaga Cunha, meu orientador, por ter aceite este encargo e pela sua disponibilidade e espírito crítico que contribuíram significativamente para a qualidade de toda a dissertação.

Aos meus colegas, Filipe Guerra e Clemente Raposo, pelo encorajamento, colaboração e amizade.

À Luísa e ao José Pinto pela disponibilidade e hospitalidade durante a minha estadia em Évora ao longo do Mestrado.

À minha namorada, Carla Pinto, pela paciência, compreensão e ajuda nesta fase final.

E, por último, mas não menos importante, aos meus pais, João Morais e Luci Selésio, e irmã, Rute Morais, pelo apoio, compreensão, diversos sacrifícios suportados e pelo constante encorajamento a fim de prosseguir não só a elaboração deste projeto, assim como, todo o meu percurso académico.

SUMÁRIO

Construtor inteligente de módulos

Criação automática de módulos para SugarCRM

As Tecnologias de Informação e Comunicação assumem, hoje em dia, um papel de extrema importância nos desenvolvimentos estratégicos e operacionais de empresas tornando a gestão de informação mais simples e eficaz.

Neste contexto surge o *CRM* (*Customer Relationship Management*), que define toda uma classe de ferramentas que automatizam as funções de contacto com o cliente, proporcionando às empresas uma melhor compreensão do comportamento dos seus clientes e a criação de ofertas de produtos e serviços personalizados.

Um exemplo prático desta aplicação é o *SugarCRM* que embora seja uma plataforma altamente extensível, apresenta algumas limitações, no que diz respeito à optimização e automatização de tarefas repetitivas sejam elas de desenvolvimento ou administração.

Assim, esta dissertação, procura desenvolver para *SugarCRM* uma alternativa, na forma de um *CLI* (*Command-Line Interface*) com o objetivo de aumentar a velocidade de desenvolvimento, ajudar na prevenção e deteção de erros e facilitar a execução de tarefas específicas.

PALAVRAS-CHAVE

Gestão de Relacionamento com o Cliente, *SugarCRM*, Interfaces de Linha de Comandos

ABSTRACT

The Information and Communication Technologies assume, nowadays, a very important role with regard to operational and strategic developments of companies.

In this context we find the CRM (Customer Relationship Management), a concept created to define a whole class of tools that automate the functions of customer contact, providing companies a better understanding of the behavior of their customers and allowing the creation of products and personalized services offerings.

A practical example application of this concept is SugarCRM, which although it is a highly extensible platform, it has some limitations, concerning the optimization and automation of repetitive tasks related with development or administration processes.

This dissertation aims to develop, an alternative, to be used by both system administrators and developers, in the form of a CLI (Command-Line Interface) for SugarCRM, that not only increases the development speed and helps preventing and detecting errors but also facilitates the execution of specific tasks.

KEYWORDS

Customer Relationship Management, SugarCRM, Command-Line Interfaces

CONTEÚDO

LISTA DE TABELAS

SIGLAS E ACRÓNIMOS

API	Application Programming Interface
CLI	Command-Line Interface
CMS	Content Management System
CRM	Customer Relationship Management
GUI	Graphical User Interface
LTS	Long Term Support
MVC	Model View Controller
RAD	Rapid Application Development
SQL	Structured Query Language
XML-RPC	Extensible Markup Language – Remote Procedure Call

CAPÍTULO 1

INTRODUÇÃO

Esta dissertação insere-se no campo da Engenharia Informática, mais precisamente na área de desenvolvimento de *software*.

Como base de investigação desta dissertação, encontram-se aplicações desenvolvidas com recurso a *SugarCRM*, que se revelam essenciais na relação entre cliente e empresa. De início optou-se por analisar todos os passos inerentes ao desenvolvimento deste tipo de aplicações e a questionar de que forma poderia ser construída uma ferramenta que ajudasse a dinamizar o processo de desenvolvimento. Desta forma, e apesar de numa primeira abordagem se ter escolhido, como foco principal, tornar mais ágil o processo de criação de novos módulos, mais tarde, e à medida que se foram consolidando conhecimentos sobre a plataforma, modificou-se o espetro de funcionalidades a desenvolver, tornando-se este bastante mais alargado.

Assim sendo, a presente dissertação espelha todo o desenvolvimento de um sistema que procura melhorar algumas das funcionalidades já existentes no *SugarCRM* que, de uma forma ou de outra, apresentam algumas limitações.

A dissertação divide-se em cinco capítulos: “Introdução”, “Trabalho relacionado”, “Sistema proposto”, “Implementação” e “Conclusão”.

Na primeira parte, capítulo um, é apresentada uma contextualização dos temas inerentes ao desenvolvimento desta dissertação de forma a tornar claro todo o seu entendimento, assim como todos os objetivos traçados para a mesma.

Em seguida, e já no capítulo dois, é apresentado o estado de arte no que diz respeito a ferramentas existentes em outros projetos e que têm em comum o conceito e metodologia adoptada no desenvolvimento da ferramenta proposta.

No terceiro capítulo, e depois de já terem sido analisadas outras interfaces e as suas características, apresenta-se o projeto a desenvolver no decorrer desta dissertação: uma ferramenta que tem por objetivo colmatar algumas das limitações existentes no *SugarCRM* enquanto plataforma de desenvolvimento e, ao mesmo tempo, possibilitar a abertura de um novo canal de desenvolvimento/administração entre o utilizador e a plataforma.

No quarto, apresenta-se a implementação do sistema proposto anteriormente; e, em quinto e último lugar, apresentam-se as conclusões de todo o projeto, assim como, objetivos para futuros desenvolvimentos.

1.1 CRM

Customer Relationship Management (CRM), em português, Gestão de Relacionamento com o Cliente, é uma metodologia que coloca o cliente no centro dos processos de negócio, desenvolvida com o intuito de perceber, antecipar e responder às necessidades dos clientes atuais e potenciais, da melhor forma.

O *CRM* apresenta-se, claramente, como uma estratégia que se materializa em soluções tecnológicas constituídas por um conjunto de procedimentos e processos bem organizados e integrados num modelo

de gestão de negócios. Estas soluções auxiliam na recolha e interpretação de dados dos clientes, nomeadamente, sobre as suas atividades e interações. Usualmente integradas na área de *marketing*, integram também as tecnologias de informação já existentes de modo a dotar a empresa de meios mais eficazes de atender, reconhecer e tratar os clientes em tempo real, com base em elementos atuais. No entanto:

> *"[...] o desafio não é somente o relacionamento de uma forma média com os clientes, mas sim um relacionamento individualizado, one-to-one (Cap Gemini, 1999) [...] O princípio base é "desenvolver e gerir relações individuais com clientes individuais" (Peppers and Rogers Group, 2000, p.5), tendo presente que os fornecedores e os clientes possuem "expectativas distintas e necessidades específicas" (Vavra, 1995, p. 15)".*[1]

Neste sentido e centrando-se essencialmente em modelos descritivos, o *CRM*, através de base de dados, procura encontrar padrões nos dados dos clientes de uma forma totalmente autónoma, de modo a conseguir um bom relacionamento entre o cliente e a empresa. Isto significa que "a empresa investe tempo e esforços para entender o que os clientes querem e não querem, e está atenta [a possíveis] alterações ao longo do tempo"[2], de forma a tornar a estratégia da empresa mais inteligente. Para além disso, o *CRM* permite também a consulta e comunicação dessas informações a diferentes setores da empresa, agilizando assim, o processo de tomada de decisões, com vista a ser criado e mantido um bom relacionamento com os seus clientes.

Tendo como foco um conjunto de processos centrados no cliente, disseminados por toda a organização, o seu objetivo principal passa por dinamizar o processo de angariação e, mais que isso, fidelização de clientes. No fundo, o "[...] objectivo é incrementar o valor a longo prazo do cliente para a organização, desenvolvendo e retendo clientes através do incremento da satisfação e lealdade (Sindell, 2000, p. 12)"[3].

Por outras palavras, o *CRM* ao permitir criar um relacionamento personalizado com cada cliente, concede às empresas a possibilidade de criação de respostas também elas personalizadas, antecipando assim, as suas vontades e respondendo de forma precisa aos seus desejos atuais. Para além disso, através da interpretação de todos os dados recolhidos, consegue fazer a identificação e a compreensão do perfil de um determinado cliente e, através dele, ter contacto com outros tantos com base num processo de referenciação, que permitirá a angariação de novos clientes.

Portanto, o *CRM*, citando José Duarte Santos, "estabelece […] um relacionamento [que] pressupõe que existe algo comum entre ambas as partes"[4]: cliente e empresa.

1.2 SUGARCRM

SugarCRM é um exemplo prático da aplicação da metodologia de *CRM*, uma plataforma corporativa desenvolvida em ambiente *Open Source* e disponível em cinco edições: *Community Edition*, *Professional*, *Corporate*, *Enterprise* e *Ultimate*.

Promovendo uma gestão integrada de informações, o *SugarCRM* disponibiliza um conjunto alargado de módulos que permitem gerir informações sobre entidades, contactos, perspectivas, oportunidades, projetos, ocorrências, campanhas de *marketing*, atividades (como telefonemas, reuniões e tarefas), entre outras. A única edição livre de licenças comerciais é a *Community Edition* que possui, por isso, um conjunto de funcionalidades inferior em relação às restantes.

> *"Whether one opts for the AGPLv3 licensed or the commercially licensed version, the application comes with all of the source code, enabling developers to customize and build upon the product with ease. This puts the control of your application and your data in your hands, enabling the freedom to deploy SugarCRM wherever you wish. This ability has given SugarCRM partners and developers worldwide the ability to customize the out-of-the-box application to fit in many different organizations, vertical markets, and locales".*[5]

Independentemente da edição, o *SugarCRM* apresenta-se como uma solução efetiva a um custo muito mais acessível, altamente flexível no que toca a configurações, com diferentes opções de instalação e de fácil integração com outras plataformas, permitindo também o desenvolvimento de módulos à medida. Por outras palavras, o *SugarCRM* apresenta e permite a criação de diferentes módulos que independentemente do contexto sobre o qual foram desenvolvidos podem interagir entre si através de relações entre os seus dados.

As inúmeras funcionalidades de personalização e configuração fazem com que seja facilmente moldável ao negócio de cada empresa, e esta é uma das suas grandes vantagens, permitir de facto aos programadores, usar esses inúmeros componentes para a construção de qualquer tipo de aplicação[6], potenciando assim uma maior adoção da plataforma face aos seus concorrentes no mercado. No entanto, embora seja uma plataforma altamente extensível e com um grande potencial, um vez que, através dela:

> *"[we] can build upon this base by customizing these modules by adding new fields, additional relationships between modules, and business logic. And [we] can take things a step further and easily design new modules using Module Builder based upon common module design templates, or a clean slate with custom-built forms and relationships. By keeping a modular design, it makes it easier to customize and build the application exactly how [we] need it, and at the same time leverage all the existing functionality the platform offers".*[7]

O *SugarCRM* apresenta algumas limitações, não só no que diz respeito à optimização e automatização de tarefas repetitivas, sejam elas de desenvolvimento ou administração, mas também no que toca à falta de uma *API* e *standards* bem definidos, o que, por conseguinte leva a que o código da plataforma, por vezes, seja dificilmente reutilizável.

1.3 INTERFACES DE LINHA DE COMANDOS

Interfaces de linha de comandos, também conhecidos por *CLIs*[1], são interfaces homem-máquina, ou seja, formas para o Homem interagir com os computadores, que visam executar uma determinada função, em modo texto, quer através da execução de comandos, quer da apresentação dos resultados que os mesmos efetuam sobre o sistema/aplicação.

> *"Many command-line interfaces provide several advantages over graphical user interfaces, still today. They are mostly popular among power users, programmers and system administrators".*[8]

Neste tipo de interfaces, os comandos (instruções que dizem para um computador fazer algo) são inseridos pelo utilizador, normalmente com recurso a um teclado, e interpretados após a tecla [*ENTER*] ser pressionada.

1 *"CLIs stand in sharp contrast to graphical user interfaces (GUIs), the other main type of human-computer interface. GUIs feature the use of graphic images, including windows, icons and menus. These objects are manipulated by a mouse (and can usually be manipulated to a limited extent by a keyboard as well)".* [9]

> *"The most familiar example of a CLI to many people is MS-DOS. However, CLIs are provided by most operating systems, although many people, particularly less skilled users, rarely, if ever, access them and may not even be aware that they exist".*[10]

Nestas interfaces cada comando suporta um conjunto muito específico de parâmetros, opcionais ou obrigatórios, o que torna a sua sintaxe bastante rígida, fazendo com que um comando só funcione caso os seus parâmetros tenham sido inseridos sem qualquer erro.

Apesar das *CLIs* apresentarem claramente vantagens e desvantagens, pois, por exemplo:

> *"There are situations in which a user will find a CLI is the most convenient and others in which the same user may prefer a GUI. Thus, the best operating systems offer users a choice of both CLIs and GUIs, thereby providing the maximum flexibility to use each where it is most advantageous or convenient".*[11]

Representam um enorme avanço no campo da informática por permitirem um controlo muito mais flexível e eficiente na construção de uma interface.

1.3.1 TIPOS DE INTERFACES DE LINHA DE COMANDOS

Existem três tipos principais de interfaces de linha de comandos [12]:

- **Interativas** - interfaces que possibilitam a interação do utilizador ao longo do seu funcionamento, levando a que o resultado das operações a executar esteja dependente dos dados introduzidos;

- **Não interativas** - interfaces que, após a sua invocação, não requerem qualquer tipo de interação do utilizador;
- **Baseadas em texto** - interfaces que se apresentam sob a forma de uma interface gráfica em modo texto por sua vez emulada num terminal, tornando-se num misto entre *GUIs* e *CLIs*.

1.3.2 VANTAGENS

Embora possa não parecer simples um utilizador ter que introduzir inúmeros comandos para realizar determinadas tarefas, é através deste tipo de interfaces que se tem uma boa compreensão do que realmente acontece aquando da nossa utilização de um computador. Em seguida enumeram-se aquelas que se consideram ser as principais vantagens das *CLIs*:

- Todas as opções e operações são apresentadas e controladas de forma consistente. Na grande maioria dos ambientes gráficos, as opções e operações disponíveis, muitas vezes aparecem dispersas por menus, níveis e padrões de uso diferentes.
- Têm a capacidade de receber comandos como um todo e sem a interação do utilizador, facilitando assim, a automação de processos muito específicos com recurso a *scripts*.
- Baixos requisitos de sistema levam a que sejam rapidamente inicializadas e que sejam regularmente usadas em sistemas integrados.

- Complexidade reduzida face a interfaces gráficas privilegia, em termos de velocidade, o seu desenvolvimento e a sua instalação em diferentes sistemas.
- Potenciam a utilização remota não só por não necessitarem de ambiente gráfico, mas também por serem menos exigentes no que diz respeito ao consumo de largura de banda.
- Apenas necessitam de teclado - outros periféricos externos, como o rato, podem ser considerados uma distração, pelo que são dispensáveis.
- São auto-documentados, isto é, para além de disponibilizarem exemplos de utilização, todos os comandos são brevemente descritos, assim como, todos os seus parâmetros.
- Potenciam não só a interação homem-máquina, mas também uma interação máquina-máquina, onde o computador substitui o humano passando assim a ser o principal responsável pela inserção de comandos.

1.3.3 DESVANTAGENS

A principal desvantagem normalmente apontada a este tipo de interfaces, é a curva de aprendizagem, que por vezes pode ser algo acentuada e obrigar à consulta de documentação sobre as mesmas. Esta dificuldade é usualmente sentida por utilizadores com pouca formação, dado que a utilização deste tipo de interfaces, regra geral, não é tão intuitiva como a de interfaces gráficas onde os utilizadores acabam por

descobrir a aplicação à medida que a vão utilizando. Para além destas existem outras, entre elas, destacam-se:

- Nem sempre haver um *feedback* adequado para cada comando. Isto significa que quando ocorre um erro o sistema apesar de não funcionar e avisar que há um erro, pode não mostrar o que está errado.
- Ao contrário do que acontece com as interfaces gráficas, estas apresentam os seus comandos com nomes que nem sempre têm significado para os utilizadores, dificultando assim, o seu entendimento e a sua identificação.

1.4 OBJETIVOS

Depois de efetuada uma análise sobre o funcionamento do *SugarCRM* de forma a perceber as suas limitações e sobre alguns dos principais *CLIs* e das suas respectivas *frameworks*, pretende-se desenvolver um conjunto de funcionalidades alternativas e/ou complementares às já existentes no *SugarCRM*, na forma de um *CLI* para *SugarCRM*. Uma ferramenta destinada tanto a programadores como administradores de sistemas, que tem como objetivo, não só aumentar a velocidade de desenvolvimento, mas também ajudar na prevenção e deteção de erros, assim como, facilitar a execução de tarefas específicas.

CAPÍTULO 2

TRABALHO RELACIONADO

2.1 ESTADO DA ARTE

Desde cedo que as interfaces de linha de comandos desempenham um papel de extrema importância no que toca à interação humana com a maioria dos sistemas operativos mais antigos, nomeadamente as primeiras versões do *MS-DOS* e *Unix*. Este tipo de interfaces recebe comandos em formato de texto e converte os mesmos em ações reconhecidas pelo programa ou sistema sobre o qual estes comandos são executados.

Ao contrário do utilizador comum que prefere interfaces gráficas, utilizadores avançados optam regularmente por usar interfaces de linha de comandos dado que fornecem formas mais concisas, rápidas e eficientes de controlar um programa ou sistema operativo.

É neste sentido que cada vez mais *frameworks* têm implementado este tipo de interfaces, procurando agilizar o processo de desenvolvimento, através de comandos que não só tornam mais célere todo este processo, como possibilitam a automação de tarefas específicas e uma rápida análise do sistema a ser implementado.

Mantendo o foco no espetro de *frameworks* de desenvolvimento orientado à *web*, foi efetuada uma pesquisa sobre as que implementam este tipo de interfaces. Dessa análise destacaram-se as seguintes interfaces de linha de comandos: *Zend Tool*, *Symfony CLI* e *Drush*, as quais analisaremos em seguida de um modo breve, realçando

as suas caraterísticas, no sentido de averiguar quais poderão vir a ser úteis aquando da realização do projeto a ser desenvolvido nesta dissertação.

2.1.1 ZEND TOOL

Disponível a partir da versão 1.8 da *Zend Framework*[2] e desenvolvida sobre uma metodologia *RAD* (*Rapid Application Development*).

> *"RAD […], is a term with a pretty wide definition. In its most general sense, its a term that describes the speed at which you can build the resources that form your application's backlog of requirements. In the most ideal of situations, the initial development, or loading phase of a project, should be kept to an absolute minimum such that the developers can get on with the more interesting development. After all, the more "interesting development" is more than likely the reason the application was born".*[13]

A *Zend Tool* surge da necessidade de agilizar todo o processo de criação de um projeto, permitindo que o programador se foque no desenvolvimento da sua ideia e das suas principais funcionalidades que, no fundo, distinguem o seu projeto dos demais, sem que para isso tenha de investir demasiado tempo na implementação de funcionalidades básicas que de certa forma são análogas a todos os projetos *web* desenvolvidos sob um padrão de desenho *MVC*.

Independentemente das capacidades e funcionalidades suportadas no âmbito de aplicações construídas com recurso a *Zend Framework*,

2 *Zend Framework, é uma framework de código aberto, publicada sob a licença New BDS e desenvolvida pela Zend Technologies, orientada ao desenvolvimento de aplicações e serviços web baseados em PHP.*

interessa-nos analisar, especificamente, a capacidade que a *Zend Tool* tem de ser utilizada/adaptada a aplicações construídas com recurso a outras *frameworks* de desenvolvimento.

Em termos de arquitetura, a *Zend Tool*, é uma ferramenta altamente extensível, onde para cada um dos seus componentes principais existe uma abstração, de forma a que não só possam ser facilmente estendidos, como possam também ser substituídos por completo. Exemplo disso é a interface inicial que, por omissão, é uma interface de linha de comandos, mas que pode ser facilmente substituída por um outro tipo de interface, *web* ou *XML-RPC*, sem que para isso seja necessário efetuar quaisquer alterações ao núcleo da aplicação propriamente dita.

Mantendo o foco no que diz respeito à interface de linha de comandos, mais especificamente na criação de comandos, a *Zend Tool* disponibiliza um conjunto de regras e abstrações que devem ser implementadas aquando da criação de novos comandos de forma a que estes sejam corretamente reconhecidos pela ferramenta.

> *"Zend_Tool finds all its commands on php's include_path. Finding which files might contain what Zend_Tool calls "Providers", is based off of some conventions set by the default Zend_Tool loader. When Zend_Tool starts up, the first thing it does is scan all the paths in your include_path in order to find files that match the regular expression ".*(?:Manifest|Provider)\.php". That essentially says any files that end in Manifest.php or Provider.php. Once it finds these files, it will require_once them into the runtime, and look for any classes within them that implement either the Zend_Tool_Framework_Manifest_Interface interface or the Zend_Tool_Framework_Provider_Interface interface. If they do indeed implement this interface, these classes are loaded into the tooling system and can now be called by your client".*[14]

Na sequência da análise efetuada conclui-se que a *Zend Tool*, sendo uma ferramenta altamente extensível, pretende dar resposta a um

conjunto de problemáticas bastante abrangente, o que faz com que embora suporte nativamente interfaces de linha de comandos, esse não seja o seu foco principal, justificando assim o conjunto reduzido não só de funcionalidades mas também de documentação que apresenta neste âmbito.

2.1.2 SYMFONY CLI

À semelhança de outras aplicações *web*, também as aplicações desenvolvidas com recurso à *Symfony Framework*[3] têm tarefas que se tornam de alguma forma repetitivas, estejam elas relacionadas com operações de manutenção, análise ou desenvolvimento. Foi com vista a agilizar este tipo de tarefas, que surgiu a *Symfony CLI*, uma interface de linha de comandos, responsável pela abertura de um novo canal de desenvolvimento/administração entre o utilizador e as aplicações desenvolvidas com recurso a esta *framework*.

Prova da importância deste tipo de interface, tem sido a constante preocupação da equipa responsável pelo desenvolvimento da *framework* em melhorar a mesma, tanto que na versão 1.1 a abstração usada para a definição de comandos foi revista e melhorada, deixando de lado as *Pake Tasks*[4], e dando origem às *Symfony Tasks*.

3 *Symfony, é uma framework de código aberto baseada em PHP, publicada sob a licença MIT e originalmente desenvolvida por Fabien Potencier, de forma a dar resposta às necessidades dos seus clientes no que toca ao desenvolvimento de aplicações e serviços web.*

4 *"Pake is a php tool similar to the Rake command, a Ruby translation of the make command. Pake was built by the symfony team. It automates some administration tasks according to a specific configuration file called pakefile.php".* [15]

> *"Symfony 1.1 introduced a modern, powerful, and flexible command line system in replacement of the old pake-based tasks system. From version to version, the tasks system has been improved to make it what it is today".*[16]

Em termos de arquitetura, a interface de linha de comandos implementada, é composta por dois componentes principais, a *Symfony CLI*, responsável pela interpretação de comandos que se refletem em ações sobre a aplicação, e a *Symfony Task*, uma poderosa e flexível abstração que põe ao dispor do programador um alargado conjunto de funcionalidades relacionadas com a definição de comandos.

No que diz respeito à caraterização de comandos, enquanto abstração, a *Symfony Task* suporta a definição de *namespaces*[5] específicos, descrição simples e detalhada do comando, e descrição tanto de argumentos obrigatórios como opcionais. Para além disso, e do ponto de vista de interação com o utilizador, disponibiliza ainda os métodos[17]:

- ask(), imprime uma pergunta e retorna o valor respondido pelo utilizador;
- askConfirmation(), imprime uma mensagem de confirmação e força a que o utilizador responda y (sim) e n (não), retornando o valor respondido;
- askAndValidate(), imprime uma pergunta e automaticamente valida o valor respondido pelo utilizador.

Após esta análise conclui-se que a *Symfony CLI* é uma ferramenta facilmente utilizada/adaptada por aplicações construídas com recurso a

5 *"Namespace is a container for a set of identifiers (names), and allows the disambiguation of homonym identifiers residing in different namespaces. Namespaces usually group names based on their functionality".* [18]

outras *frameworks* de desenvolvimento. O facto de ser altamente extensível faz com que a forma como é implementada sobre a aplicação em causa seja deixada ao critério do programador. Para além disso, são ainda de destacar as melhorias efetuadas a cada nova versão da *Symfony framework* não só em termos de código mas também de documentação, o que revela a importância deste componente e a forma ativa como tem vindo a ser desenvolvido. Por outro lado a descontinuação e constante reformulação do mesmo, demonstram que ainda não se encontra suficientemente maduro para que possa ser usado em projetos *LTS*.

2.1.3 DRUSH

O *Drush*[6] é uma interface de linha de comandos, publicada sob a licença GNU GPL v2, para *Drupal*[7]. E, ainda que o *Drupal* possa ser considerado uma *framework*, é na realidade um *CMS* (*Content Management System*) altamente flexível, que tal como as *frameworks* descritas atrás, sofre do mesmo tipo de problemáticas no que toca a repetição de tarefas relacionadas com a construção e manutenção de aplicações *web*.

6 *"Drush is a command line shell and scripting interface for Drupal, a veritable Swiss Army knife designed to make life easier for those of us who spend some of our working hours hacking away at the command prompt".* [19]

7 *Drupal é uma framework de código aberto baseada em PHP, originalmente desenvolvida por Dries Buytaert, e que conta com uma comunidade ativa de mais de 630.000 utilizadores.*

> *"Most of the tasks for building and maintaining a website are repetitive and involve filling in forms on administration pages. The majority of these tasks can be achieved with a single Drush command, drastically shortening the development and maintenance time of a project".*[20]

Ao contrário das interfaces de linha de comandos anteriormente apresentadas, o *Drush* não é incluído no pacote de instalação do *Drupal*, mas disponibilizado em separado como um utilitário e para além disso, também não é desenvolvido pela equipa responsável pelo *Drupal*, mas pela comunidade.

No que toca à arquitetura, também esta é diferente das anteriormente apresentadas, não só porque se baseia em *hooks*[8]*, um* conceito também utilizado no *Drupal*, mas também devido a outras particularidades que o distinguem das demais, nomeadamente o conceito de *bootstrapping*, útil aquando da definição de comandos. No decorrer da execução de um comando, o *Drush* despoleta um processo de inicialização do ambiente *Drupal* em várias fases. Esta inicialização faseada, *bootstrapping*, permite que a configuração de comandos seja bastante mais abrangente, criando assim a possibilidade de definir dependências relativamente a estas diferentes fases. Na especificação de um comando, para além de um conjunto bastante alargado de propriedades[9], pode ser definida uma fase, tornando-o assim dependente da mesma, ou seja, esta indicará qual a fase mínima de inicialização do ambientes *Drupal* a ser atingida para que o comando possa ou não ser executado.

8 *"[…] the term hooking covers a range of techniques used to alter or augment the behavior […] of applications, […] by intercepting function calls or messages or events passed between software components".*[22]

9 *Conjunto de propriedades suportadas por um comando disponível em: "http://www.drush.org/docs/commands.html".*

No que diz respeito à criação de novos comandos é disponibilizado um conjunto de regras que deve ser respeitado, de forma a que estes sejam corretamente reconhecidos pela ferramenta[21]:

- A file named COMMANDFILE.drush.inc where COMMANDFILE is the namespace of the group of commands that will be implemented
- An implementation of the hook COMMANDFILE_drush_help() which optionally describes each command and how they are categorized in the output of the drush help command
- An implementation of the hook COMMANDFILE_drush_command() where the basic properties of each command are defined
- A callback function for each defined command at COMMANDFILE_drush_ command() that will do the actual processing following the function name drush_COMMANDFILE_COMMANDNAME".

Para além disso, e do ponto de vista de interação com o utilizador, disponibiliza ainda os seguintes métodos[23]:

- drush_choice() imprime uma lista de opções e força a que o utilizador selecione uma, retornando a opção selecionada.
- drush_choice_multiple() imprime uma lista de opções e força a que o utilizador seleccione pelo menos uma, retornando o conjunto de opções selecionadas.
- drush_confirm() imprime uma mensagem de confirmação e força a que o utilizador responda y (sim) e n (não), retornando o valor

respondido;

- drush_prompt() imprime uma pergunta e retorna o valor respondido pelo utilizador.

Na sequência da análise efetuada, conclui-se que o *Drush* é uma interfaces de linha de comandos, altamente extensível e configurável, que pode ser facilmente adaptada à realidade de aplicações construídas com recurso a outras *frameworks* de desenvolvimento. Para além disso é de destacar a evolução desta ferramenta, que, mesmo não sendo parte integrante do pacote de instalação do *Drupal*, tem acompanhado o desenvolvimento do mesmo enquanto plataforma, assim como, para além das particularidades já apresentadas que fazem com que sobressaia perante as demais, a quantidade e qualidade de documentação existente, e a vasta e ativa comunidade.

CAPÍTULO 3

SISTEMA PROPOSTO

3.1 APRESENTAÇÃO DO SISTEMA

Depois de terem sido analisadas diferentes interfaces de linhas de comandos (Capítulo 2) de modo a clarificar o modo de funcionamento de cada uma delas e a perceber, depois, qual delas se adequava melhor ao objetivo pretendido nesta dissertação, apresenta-se, agora, de forma mais clara todo o projeto.

Antes de passar à implementação propriamente dita, a primeira decisão passou por perceber de que forma faria sentido avançar com o desenvolvimento, isto significa que, antecipadamente, foi necessário decidir entre desenvolver algo de raiz ou recorrer a uma outra ferramenta já existente que pudesse ser adaptada à realidade do *SugarCRM*.

Criar algo de raiz, para além de ter um custo elevado em termos temporais, nem sempre se justifica, especialmente se já existirem ferramentas semelhantes que possam ser reutilizadas e/ou adaptadas de forma a servir um propósito específico, permitindo assim que o foco se concentre na implementação de novas funcionalidades.

Com isto em mente e após a análise efetuada (no Capítulo 2), optou-se por utilizar o *Drush* como *framework* de desenvolvimento, não só pelo facto de ser uma ferramenta criada com um propósito muito específico, semelhante ao que pretendemos atingir com este projeto, mas também por ser altamente extensível, configurável e por apresentar particularidades (Capítulo 2), nomeadamente o conceito de *boostrapping*, que farão sentido integrar neste projeto, tendo em conta o funcionamento do *SugarCRM*.

Assim sendo, na sequência dos objetivos apresentados (no Capítulo 1) e à semelhança daquilo que o *Drush* representa enquanto interface de linha de comandos, pretende-se não só proceder à adaptação deste, àquele que é o funcionamento do *SugarCRM* mas também proceder ao desenvolvimento de um conjunto específico de comandos que permitam não só aumentar a velocidade de desenvolvimento, mas também ajudar na prevenção e deteção de erros, assim como, facilitar a execução de tarefas específicas.

3.2 FERRAMENTAS A UTILIZAR

À semelhança do *SugarCRM* também o *Drush* é desenvolvido com recurso a tecnologias *Open Source*, nomeadamente com *PHP*, pelo que os desenvolvimentos a efetuados também o foram, com especial foco em versões iguais ou superiores à versão 5.2.

Relativamente ao desenvolvimento propriamente dito, foi usado *Git* como sistema de controlo de versões, com recurso ao *Unfuddle* - um serviço *web* de alojamento de projetos com sistemas de controlo de

versões, que permite que o projeto seja mantido como privado de forma gratuita.

Posteriormente, após apresentação e avaliação desta dissertação, o resultado final será realojado no *github*, um serviço semelhante ao *Unfuddle*, mas exclusivamente dedicado a projetos de desenvolvimento de *software* que usam *Git* como sistema de controlo de versões.

CAPÍTULO 4

IMPLEMENTAÇÃO

4.1 INSULIN - 'CAUSE TOO MUCH SUGAR CAUSES DIABETES!'

Tomada a decisão de usar *Drush* como *framework* de desenvolvimento, surge a necessidade de criar um nome para a ferramenta, de forma a identificá-la perante as demais.

Assim, de um trocadilho, resulta o nome "*Insulin – 'Cause too much Sugar causes diabetes!"*, uma interface de linha de comandos que resulta da adaptação de *Drush* à realidade de *SugarCRM*.

4.2 COMANDOS

4.2.1 ESPECIFICAÇÕES

Como indicado no Capítulo 2, no *Drush*, a especificação de comandos obedece a um determinado conjunto de regras, que compreendem não só a nomenclatura de ficheiros, comandos e funções, como também as propriedades suportadas e a localização dos mesmos no que diz respeito ao sistema de ficheiros. Em seguida irá ser analisado o conjunto de regras adaptadas.

4.2.1.1 NOMENCLATURA DE FICHEIROS, COMANDOS E FUNÇÕES

À semelhança do que é feito no *Drush*, também na *Insulin*, o nome dos ficheiros que contêm definições de comandos é identificado com um sufixo específico, neste caso ".insulin.inc", de modo a ser reconhecido como tal. De igual forma, o texto que o precede, identifica o grupo no qual os comandos estão inseridos.

Exemplo:

Ficheiro: user.insulin.inc

Isto significa que o ficheiro "user.insulin.inc", contém documentação relacionada com o grupo de comandos "user" e as definições dos comandos nele inseridos.

Relativamente ao nome de cada comando, este deve ser precedido do nome do grupo em que está inserido, "<grupo>-<comando>", assim como, o nome da função responsável por definir a lógica de negócio deste, deve ter o prefixo "insulin_" agregado ao nome do grupo e do respetivo comando, isto é, "insulin_<grupo>_<comando>".

Exemplo:

Grupo: user

Comando: user-information

Função: insulin_user_information

4.2.1.2 LOCALIZAÇÃO

Relativamente aos comandos que fazem parte do núcleo da *Insulin*, estes devem seguir as regras indicadas no tópico anterior (4.2.1.1) no que diz respeito à nomenclatura de ficheiros. Para que sejam corretamente reconhecidos, devem também ser definidos no diretório: "insulin/commands". Já no que diz respeito a outros comandos que possam ser criados como extensão ao que é originalmente implementado, estes podem ser definidos em diferentes diretórios, de acordo com a lista, ordenada por precedência, apresentada na Tabela 4.1. Ainda em relação à nomenclatura dos ficheiros contidos nestes últimos, são reconhecidos os ficheiros "insulinrc.php" e "insulin<version>rc.php" onde <*version*> diz respeito à versão da *Insulin* a que os comandos se destinam.

Tabela 4.1: Diretórios de comandos e respetivas precedências

PRECEDÊNCIA	DIRETÓRIO
1	/sugar/custom/insulin
2	Diretório específico definido pelo utilizador com recurso ao parâmetro --config.
3	~/.insulin
4	/etc/insulin
5	/insulin/commands

4.2.2 BOOTSTRAPS

Como mencionado anteriormente (Capítulo 2), aquando da execução de um comando, o *Drush* despoleta um processo de inicialização, *bootstrapping*, do ambiente *Drupal* em várias fases, Tabela 4.2, à semelhança do que é efetuado quando é realizado um pedido a uma aplicação desenvolvida com a mesma plataforma a correr num servidor *web*; como referido na documentação [24]:

> *"For efficiency and convenience, some drush commands can work without first bootstrapping a Drupal site, or by only partially bootstrapping a site. This is more efficient, because there is sometimes a slight delay involved with bootstrapping, especially in some of the later stages. It is also a matter of convenience, because some commands are useful to use even when you do not have a working Drupal site available to bootstrap. For example, [we] can use drush to download Drupal with `drush dl drupal`. This obviously does not require any bootstrapping to work".*

Em suma, para que um comando possa ser executado, não é obrigatória a inicialização completa, ou parcial do ambiente *Drupal*.

Tabela 4.2: Fases de inicialização do *Drush*

FASE	DESCRIÇÃO
DRUSH_BOOTSTRAP_DRUSH	Inicialização de variáveis que dependem apenas e exclusivamente do *Drush*.
DRUSH_BOOTSTRAP_DRUPAL_ROOT	Verificação da validade do ambiente *Drupal*, recorrendo para isso a uma avaliação sobre o diretório onde se encontra uma instância. Aquando da execução do comando em questão, pode ser definido um parâmetro onde é indicado qual o diretório onde se encontra a instância, caso este parâmetro não seja indicado, o diretório actual será usado para esse efeito.

Tabela 4.2: Fases de inicialização do *Drush* (continuação)

FASE	DESCRIÇÃO
DRUSH_BOOTSTRAP_DRUPAL_SITE	Inicialização de variáveis dependentes de um *site* em específico. Aquando da execução do comando em questão, pode ser definido um parâmetro onde é indicado qual o *site* a ser instanciado, caso este parâmetro não seja indicado, o *site* '*default*' será usado para esse efeito.
DRUSH_BOOTSTRAP_DRUPAL_CONFIGURATION	Inicialização de variáveis dependentes do ambiente, inclusive as de configuração. Este é também o primeiro nível onde o código da instância definida fica acessível.
DRUSH_BOOTSTRAP_DRUPAL_DATABASE	Conexão à base de dados, usando para isso as configurações carregadas na fase anterior.
DRUSH_BOOTSTRAP_DRUPAL_FULL	Não é realmente uma fase, mas sim um indicador que permite a definição de comandos que apenas podem ser executados após o ambiente ter sido inicializado por completo.
DRUSH_BOOTSTRAP_DRUPAL_LOGIN	Autenticação de um utilizador, por defeito a autenticação é efetuada com um utilizador considerado "anónimo", dado que tem acesso limitado às funcionalidades existentes. Em alternativa, com base num parâmetro específico, pode ser efetuada a autenticação com outro utilizador.
DRUSH_BOOTSTRAP_MAX	Não é realmente uma fase, mas sim um indicador que permite a definição de comandos que podem ser executados independentemente da fase de inicialização em questão, podendo, no entanto, disponibilizar informações adicionais tendo em conta a fase em que se encontram.

Após uma análise cuidada sobre as várias fases de inicialização existentes no *Drush*, Tabela 4.2, procedeu-se à adaptação das mesmas para a *Insulin*, Tabela 4.3., tendo em conta a forma como o ambiente *SugarCRM* é inicializado.

Tabela 4.3: Fases de inicialização da *Insulin*

FASE	DESCRIÇÃO
INSULIN_BOOTSTRAP_INSULIN	Inicialização de variáveis que dependem apenas e exclusivamente da *Insulin*.
INSULIN_BOOTSTRAP_SUGAR_ROOT	Verificação da validade do ambiente *SugarCRM*, recorrendo para isso a uma avaliação sobre o diretório onde se encontra uma instância. Aquando da execução do comando em questão, pode ser definido um parâmetro onde é indicado qual o diretório onde se encontra a instância, caso este parâmetro não seja indicado, o diretório actual será usado para esse efeito. No que diz respeito à validade do ambiente propriamente dito, nesta fase é avaliada a presença de três ficheiros que permitem verificar a existência de uma instância de *SugarCRM* no diretório em questão: 1. include/entryPoint.php 2. include/MVC/SugarApplication.php 3. config.php
INSULIN_BOOTSTRAP_SUGAR_CONFIGURATION	Inicialização das variáveis dependentes do ambiente, inclusive as de configuração. Este é também o primeiro nível onde o código da instância definida fica acessível.
INSULIN_BOOTSTRAP_SUGAR_DATABASE	Conexão à base de dados, usando para isso as configurações carregadas na fase anterior.
INSULIN_BOOTSTRAP_SUGAR_FULL	Não é realmente uma fase, mas sim um indicador que permite a definição de comandos que apenas podem ser executados após o ambiente ter sido inicializado por completo.
INSULIN_BOOTSTRAP_SUGAR_LOGIN	Autenticação de um utilizador, por defeito a autenticação é efetuada com o utilizador de sistema. Em alternativa, com base num parâmetro específico, pode ser efetuada a autenticação com outro utilizador.
INSULIN_BOOTSTRAP_MAX	Não é realmente uma fase, mas sim um indicador que permite a definição de comandos que podem ser executados independentemente da fase de inicialização em questão, podendo no entanto, disponibilizar informações adicionais tendo em conta a fase em que se encontram.

4.2.3 ANATOMIA DE UM COMANDO

Após breve descrição do conjunto de regras adaptadas analisamos em seguida a anatomia de um comando no que diz respeito às suas definições e implementação, tendo como exemplo o comando *field-list,* Anexo 5.

De acordo com o descrito nos Tópicos 4.2.1.1 e 4.2.1.2, este comando é definido no ficheiro *field.insulin.inc*, fazendo assim parte do conjunto de comandos associados ao grupo *field.* Em relação à sua localização em termos de sistema de ficheiros, o ficheiro reside no diretório */insulin/commands*. Relativamente ao conteúdo, ao processo de implementação e ao reconhecimento de comandos é de salientar a definição de duas funções: a função *field_insulin_command*, responsável pelo conjunto de comandos disponíveis no grupo e suas respetivas caraterísticas (Tabela 4.4), e a função *insulin_field_list*, responsável pela lógica de negócio do comando em questão.

Tabela 4.4: Caraterísticas do comando *field-list*

PARÂMETRO	DESCRIÇÃO
aliases	Conjunto de atalhos suportados pelo comando, sobre os quais o comando pode ser invocado.
	flst
arguments	Conjunto de argumentos suportados pelo comando, de notar que estes valores são de preenchimento obrigatório.
	modules

Tabela 4.4: Caraterísticas do comando *field-list* (continuação)

PARÂMETRO	DESCRIÇÃO
aliases	Conjunto de atalhos suportados pelo comando, sobre os quais o comando pode ser invocado. flst
arguments	Conjunto de argumentos suportados pelo comando, de notar que estes valores são de preenchimento obrigatório. modules
bootstrap	Fase de inicialização do ambiente *SugarCRM* sobre a qual a execução do comando está dependente, Tabela 4.3. O comando em questão depende não só da validade do ambiente sobre o qual irá interagir como também depende das suas variáveis de configuração e de acesso ao código da instância propriamente dito. INSULIN_BOOTSTRAP_SUGAR_CONFIGURATION
description	Descrição do comando. Print information about fields declared in module(s).
examples	Exemplos de utilização do comando. insulin field-list Accounts insulin field-list Accounts –type=enum insulin field-list Accounts –type=enum,int insulin field-list Accounts –source=db insulin field-list Accounts --print
options	Conjunto de opções suportados pelo comando, podendo ser usadas a solo ou em conjunto, cada uma com um propósito específico. A sua utilização influencia o resultado obtido. pipe, print, type

4.3 COMANDOS IMPLEMENTADOS

Cada comando, Tabela 4.5, para além de suportar a definição de dependências relativamente a que fase de inicialização, Tabela 4.3, pode ser executado, tem também um conjunto muito específico de parâmetros obrigatórios e opcionais, cujo preenchimento não só deve ser respeitado, como os valores passados a estes, devem também ser válidos no contexto em que são inseridos (Capítulo 2).

Relativamente aos parâmetros opcionais, estes podem ser locais, com especificidades próprias definidas pelo comando, ou globais (Anexo 2, Exemplo 1/2), suportados de forma opcional por cada comando e com especificidades próprias definidas pela interface de linha de comandos.

À semelhança do funcionamento de outras interfaces de linha de comandos (Capítulo 1), também estes comandos são auto-documentados, isto é, para além de disponibilizarem exemplos de utilização, todos são brevemente descritos, assim como, todos os seus parâmetros. Além disso cada comando suporta também a definição de abreviaturas, permitindo assim que o mesmo possa ser utilizado sem que para isso seja necessário escrever o seu nome por completo – esta funcionalidade é particularmente útil quando o nome de um comando é demasiado extenso.

Tabela 4.5: Comandos Implementados

COMANDO	DESCRIÇÃO	ANEXO
`insulin`	Exibição de lista de parâmetros globais e lista de comandos disponíveis agrupados por tema e respetivas descrições.	1
`help`	Ajuda contextualizada, com base num comando ou abreviatura, onde são exibidas informações relativas ao mesmo, nomeadamente uma breve descrição, exemplos de funcionamento, parâmetros obrigatórios/opcionais e abreviaturas. Por omissão, é apresentada lista de parâmetros globais e lista de comandos disponíveis agrupados por tema e respetivas descrições.	2
`core-quick-repair`	Reparação e reconstrução do sistema.	3
`cron-run`	Execução de eventos calendarizados.	4
`field-list`	Exibição de lista de campos de um ou mais módulos.	5
`field-missing-in-bean`	Exibição de lista de campos de um ou mais módulos, que não estejam definidos como propriedades das classes que representam os mesmos.	6
`field-missing-in-database`	Exibição de lista de campos de um ou mais módulos, que não estejam definidos nas tabelas da base de dados, que representa a estrutura dos mesmos.	7
`field-missing-in-vardefs`	Exibição de lista de campos de um ou mais módulos, que não estejam definidos nas *vardefs* dos mesmos.	8
`role-list`	Exibição de lista de funções de utilizadores.	9
`sugar-version`	Exibição da edição, da versão e do número da *build* da instância de *Sugar* a ser utilizada.	10
`team-list`	Exibição de lista de equipas.	11

Tabela 4.5: Comandos implementados (continuação)

COMANDO	DESCRIÇÃO	ANEXO
`user-create`	Criação de contas de utilizadores.	12
`user-information`	Exibição de informações de contas de utilizadores, através do Identificador de conta, nome de utilizador ou *email*. Os detalhes a exibir podem ser apresentados como um todo, ou de acordo com um conjunto de grupos de detalhe.	13
`version`	Exibição da versão da *insulin*.	14

4.4 CASOS PRÁTICOS DE UTILIZAÇÃO

No que diz respeito à utilização real da ferramenta desenvolvida, esta pode servir diferentes propósitos, não só em termos de desenvolvimento:

- auxiliando na **prevenção e deteção de erros**, através da análise bidirecional de sincronização entre campos declarados num módulo em específico e da tabela de base de dados desse mesmo módulo, através dos comandos *field-missing-in-database* (Anexo 7) e *field-missing-in-vardefs* (Anexo 8)*;*
- facilitando a definição de processos de **integração contínua**, através do comando *core-quick-repair* (Anexo 3), garantindo assim a automatização do processo, algo que até ao momento não era possível sem recorrer diretamente à plataforma.

mas também no caso da administração da plataforma, através da:

- **facilidade de execução de tarefas**, através dos comandos *user-*

create (Anexo 12) e *user-information* (Anexo 13), é permitida a criação de novas contas de utilizadores e consulta de contas já existentes, respetivamente, sem que para isso seja necessário aceder diretamente à plataforma.

- **automatização de tarefas**, através dos comandos *role-list* (Anexo 9) e team-*list* (Anexo 11) tornasse bastante mais simples a exportação de um conjunto específico de dados em formato *SQL*, mediante os parâmetros inseridos.

CAPÍTULO 5

CONCLUSÕES E TRABALHO FUTURO

A presente dissertação propõe a abertura de um novo canal de desenvolvimento/administração entre o utilizador e o *SugarCRM*, com origem numa ferramenta, proposta na sua génese como uma interface de linha de comandos, implementada sobre *Drush*. No entanto, dado que *Drush* é uma ferramenta específica que permite a interação com *Drupal*, foi necessária a sua adaptação à realidade de *SugarCRM*, (Capítulo 4), de forma a que esta fosse não só capaz de interpretar as caraterísticas do ambiente *SugarCRM* mas também de atuar sobre o mesmo.

Em todo o processo foram produzidas sensivelmente 10.000 linhas de código, tanto na adaptação como também na implementação de novas funcionalidades.

Uma das dificuldades sentidas no desenvolvimento dos comandos, (Capítulo 4), está diretamente relacionada com a falta de uma *API* e *standards* bem definidos no *SugarCRM*, do ponto de vista do desenvolvimento, o que, por vezes, levou à re-escrita de algumas funcionalidades do núcleo da plataforma de modo a criar mecanismos de extensibilidade e permitir a implementação de alguns destes comandos.

Para além do trabalho desenvolvido, no sentido de reportar à *SugarCRM* não só estas limitações, assim como também contribuir com correções, pedidos de novas funcionalidades ou alterações sobre as já existentes, pretende-se que de futuro estas interações se tornem cada vez mais regulares de forma a manter uma participação ativa na comunidade e no desenvolvimento do núcleo da plataforma.

Relativamente à ferramenta implementada, dada a sua flexibilidade, pretende-se no futuro não só desenvolver novos comandos, com o intuito de facilitar e dinamizar as interações com a plataforma, mas também, manter os comandos já existentes atualizados de acordo com as novas versões de *SugarCRM*.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Santos, José Duarte; “CRM offline & online”. Vila Nova De Gaia:Instituto Superior Politécnico Gaya. 2006. p.5

[2] *Idem;* p.21

[3] *Idem;* p.5

[4] *Idem;* p.21

[5] Mertic, John; “Building on SugarCRM”. Sebastopol, CA: O'Reilly Media. 2011. p.1

[6] *Idem*

[7] *Idem;* p.4

[8] AntoArts. "Designing command-line interfaces”. Disponível em <URL: *http://www.antoarts.com/designing-command-line-interfaces/* >

[9] Linfo - The Linux Information Project. "Command Line Interface Definition”. Disponível em <URL: *http://www.linfo.org/command_line_interface.html*>

[10] *Idem*

[11] *Idem*

[12] AntoArts. "Designing command-line interfaces”. Disponível em <URL: *http://www.antoarts.com/designing-command-line-interfaces/* >

[13] Schindler, Ralph; “Zend_Tool and ZF 1.8 - So what is Zend_Tool?”. Disponível em <URL: *http://devzone.zend.com/1451/zend_tool-and-zf-18/*>

[14] Schindler, Ralph; “Zend_Tool for the Developer”. Disponível em <URL: *http://devzone.zend.com/1349/zend_tool-for-the-developer/*>

[15] Symfony – Open-Source PHP Web Framework; “The symfony Cookbook – Command Line Interface”. Disponívem em <URL: *http://www.symfony-project.org/cookbook/1_0/en/cli*>

[16] Symfony – Open-Source PHP Web Framework; “The More with symfony book – Leveraging the Power of the Comand Line”. Disponível em <URL: *http://www.symfony-project.org/more-with-symfony/1_4/en/13-Leveraging-the-Power-of-the-Command-Line*>

[17] Symfony – Open-Source PHP Web Framework; “The More with symfony book – Leveraging the Power of the Comand Line”. Disponível em <URL: *http://www.symfony-project.org/more-with-symfony/1_4/en/13-Leveraging-the-Power-of-the-Command-Line* >

[18] Wikipédia; "Namespace”. Disponível em <URL: http://en.wikipedia.org/wiki/Namespace >

[19] WeitzMan, Moshe; “Drush”. Disponível em <URL: *http://drupal.org/project/drush* >

[20] Requena, Juan Pablo Novillo; “Drush User's Guide: A practical guide to Drush, Drupal's command line interface, helping you work with your Drupal sites more effectively”. Birmingham: Packt Publishing. 2012. p.12

[21] *Idem*; p.71

[22] Wikipédia; "Hooking”. Disponível em <URL: *http://en.wikipedia.org/wiki/Hooking* >

[23] Drush API; “Functions & methods”. Disponível em <URL: *http://api.drush.org/api/drush/includes%21drush.inc/group/userinput/* >

[24] Drush: A command line shell and scripting interface for Drupal. “The Drush Bootstrap Process”. Disponível em <URL: *http://www.drush.org/docs/bootstrap.html*>

ANEXOS

ANEXO 1. INSULIN

COMANDO:

```
[:~$] insulin
```

EXEMPLO (1/1):

```
[:~$] insulin

Execute an insulin command. Run `insulin help [command]` to view command-specific help.

Global options:
 -d, --debug                                          Display even more information, including internal messages.
 -h, --help                                           This help system.
 -n, --no                                             Assume 'no' as answer to all prompts.
 --php=<path/to/file>                                 The absolute path to your PHP intepreter, if not 'php' in the path.
 -p, --pipe                                           Emit a compact representation of the command for scripting.
 -r <path>, --root=<path>                             Sugar root directory to use (default: current directory).
 -s, --simulate                                       Simulate all relevant actions (don't actually change the system).
 -v, --verbose                                        Display extra information about the command.
 --version                                            Show insulin version.
 -y, --yes                                            Assume 'yes' as answer to all prompts.

Core insulin commands: (core)
 help                                                 Print this help message. See `insulin help help` for more options.
 core-quick-repair (repair)                           Repairs and rebuilds DB, Extensions, Vardefs, Sugar Dashlets etc..
 cron-run                                             Run system cron jobs.
 sugar-version (sversion)                             Print Sugar flavor, version and build number.
 version                                              Print insulin version.

Field commands: (field)
 field-list                                           Print information about fields declared in module(s).
 field-missing-in-bean                                Print information about fields declared in module(s) vardefs but not
                                                      defined as bean properties.
 field-missing-in-database                            Print information about fields declared in module(s) vardefs but not
                                                      defined in respective database table(s).
 field-missing-in-vardefs                             Print information about fields declared in module(s) database table but
                                                      not defined in respective vardefs.
```

```
Role commands: (role)
 role-list                                          Print information about system role(s).

Team commands: (team)
 team-list                                          Print information about system team(s).

User commands: (user)
 user-create (ucrt)                                 Create a new user account.
 user-information (uinf)                            Print information about the specified user(s).
```

ANEXO 2. HELP

COMANDO:

```
[:~$] insulin help
```

EXEMPLO (1/1):

```
[:~$] insulin help

Execute an insulin command. Run `insulin help [command]` to view command-specific help.

Global options:
 -d, --debug                                          Display even more information, including internal messages.
 -h, --help                                           This help system.
 -n, --no                                             Assume 'no' as answer to all prompts.
 --php=<path/to/file>                                 The absolute path to your PHP intepreter, if not 'php' in the path.
 -p, --pipe                                           Emit a compact representation of the command for scripting.
 -r <path>, --root=<path>                             Sugar root directory to use (default: current directory).
 -s, --simulate                                       Simulate all relevant actions (don't actually change the system).
 -v, --verbose                                        Display extra information about the command.
 --version                                            Show insulin version.
 -y, --yes                                            Assume 'yes' as answer to all prompts.

Core insulin commands: (core)
 help                                                 Print this help message. See `insulin help help` for more options.
 core-quick-repair (repair)                           Repairs and rebuilds DB, Extensions, Vardefs, Sugar Dashlets etc..
 cron-run                                             Run system cron jobs.
 sugar-version (sversion)                             Print Sugar flavor, version and build number.
 version                                              Print insulin version.

Field commands: (field)
 field-list                                           Print information about fields declared in module(s).
 field-missing-in-bean                                Print information about fields declared in module(s) vardefs but not
                                                      defined as bean properties.
 field-missing-in-database                            Print information about fields declared in module(s) vardefs but not
                                                      defined in respective database table(s).
 field-missing-in-vardefs                             Print information about fields declared in module(s) database table but
                                                      not defined in respective vardefs.
```

```
Role commands: (role)
 role-list                                          Print information about system role(s).

Team commands: (team)
 team-list                                          Print information about system team(s).

User commands: (user)
 user-create (ucrt)                                 Create a new user account.
 user-information (uinf)                            Print information about the specified user(s).
```

ANEXO 3. CORE-QUICK-REPAIR

COMANDO:

```
[:~$] insulin help core-quick-repair
```

EXEMPLO (1/2):

```
[:~$] insulin help core-quick-repair

Repairs and rebuilds DB, Extensions, Vardefs, Sugar Dashlets etc..

Examples:
 insulin core-quick-repair                                    Repairs and rebuilds DB, Extensions, Vardefs, Sugar Dashlets etc..
 insulin core-quick-repair --clearVardefs                     Repairs and rebuilds Vardefs for all modules.
 insulin core-quick-repair --modules=Account --clearVardefs   Repairs and rebuilds Vardefs for Account module.

Options:
 --modules                                                    A comma delimited list of module names, or bean names.
 --clearTpls                                                  Clear Smarty templates from cache.
 --clearJsFiles                                               Clear Javascript files.
 --clearDashlets                                              Clear Sugar Dashlet files.
 --clearThemeCache                                            Clear Theme Cache.
 --clearVardefs                                               Clear Vardefs from cache.
 --clearJsLangFiles                                           Clear Javascript Language files from cache.
 --execute                                                    Automatically applies changes between database and vardefs.
 --clearSearchCache                                           Clear Unified Search Cache.
 --clearPDFFontCache                                          Clear PDF Font Cache File.
 --rebuildAuditTables                                         Rebuild Audit Tables.
 --clearAll                                                   Clear all of the above (default).
 --rebuildExtensions                                          Rebuild Extensions.
 --repairDatabase                                             Syncs Database tables with vardefs.
 --repairRoles                                                Repairs Roles.

Aliases: repair
```

EXEMPLO (2/2):

```
[:~$] insulin core-quick-repair
```

```
Repair and Rebuild successfully.                                                [success]
Changes found between database and vardefs, what do you want to do?

 [0]  :  Cancel
 [1]  :  Print changes
 [2]  :  Apply all changes
 [3]  :  Apply changes one by one
1
ALTER TABLE leads   add column do_not_call bool  DEFAULT '0' NULL ,  add column birthdate date  NULL;
ALTER TABLE accounts   add column skype varchar(100)  NULL;

What do you want to do?
 [0]  :  Cancel
 [1]  :  Apply all changes
 [2]  :  Apply changes one by one
1
Database changes applied.                                                       [success]
```

ANEXO 4. CRON-RUN

COMANDO:

```
[:~$] insulin help cron-run
```

EXEMPLO (1/1):

```
[:~$] insulin help cron-run

Run system cron jobs.

Examples:
 insulin cron-run                                  Runs all active jobs who are able to be executed according to their last
                                                   run time.
 insulin cron-run --force                          Runs all jobs without checking last run time.
 insulin cron-run --jobs=1,2,3                     Runs cron for job with Ids 1, 2 and 3.

Options:
 --jobs                                            A comma delimited list of job ids.
 --force                                           Force jobs to run without checking last run time.

Aliases: crun
```

ANEXO 5. FIELD-LIST

COMANDO:

```
[:~$] insulin help field-list
```

EXEMPLO (1/1):

```
[:~$] insulin help field-list


Print information about fields declared in module(s).

Examples:
 insulin field-list Accounts                          Print number of fields found in Accounts module.
 insulin field-list Accounts --type=enum              Print number of fields of type enum found in Accounts module.
 insulin field-list Accounts --type=enum,int          Print number of missing properties related with fields of type enum
                                                      or int found in Accounts module.
 insulin field-list Accounts --source=db              Print number of fields of source db found in Accounts module.
 insulin field-list Accounts --print                  Print list fields found in Accounts module.

Arguments:
 modules                                              A comma delimited list of module names, or bean names.

Options:
 --pipe                                               Print a compact representation of  the command for scripting.
 --print=<tabular>                                    Print the list of fields.
 --type                                               A comma delimited list of field types.
 --source                                             The fields source. Choices: 'db', 'non-db'.


Aliases: flst
```

ANEXO 6. FIELD-MISSING-IN-BEAN

COMANDO:

```
[:~$] insulin field-missing-in-bean
```

EXEMPLO (1/5):

```
[:~$] insulin help field-missing-in-bean

Print information about fields declared in module(s) vardefs but not defined as bean properties.

Examples:
 insulin field-missing-in-bean Accounts                    Print number of missing properties found in Accounts module.
 insulin field-missing-in-bean Accounts --type=enum        Print number of missing properties related with fields of type enum
                                                           found in Accounts module.
 insulin field-missing-in-bean Accounts --type=enum,int    Print number of missing properties related with fields of type enum or
                                                           int found in Accounts module.
 insulin field-missing-in-bean Accounts --source=db        Print number of missing properties related with fields of source db
                                                           found in Accounts module.
 insulin field-missing-in-bean Accounts --print            Print list of missing properties found in Accounts module.
 insulin field-missing-in-bean accounts --print=php        Print in PHP format the list of missing properties found in Accounts
                                                           module.


Arguments:
 modules                                                   A comma delimited list of module names, or bean names.

Options:
 --pipe                                                    Print a compact representation of the command for scripting.
 --print=<tabular>                                         Print the list of missing properties. Choices: 'tabular' (default),'php'.
 --type                                                    A comma delimited list of field types.
 --source                                                  The fields source. Choices: 'db', 'non-db'.
```

EXEMPLO (2/5):

```
[:~$] insulin field-missing-in-bean Leads --type=url,bool,varchar --print

Found 9 missing properties in 'Leads' bean.                                    [success]

#           name                                                    type            source
1.          accepted_status_id                                      varchar         non-db
2.          accepted_status_name                                    enum            non-db
3.          assistant                                               varchar         db
4.          converted                                               bool            db
5.          email_opt_out                                           bool            non-db
6.          invalid_email                                           bool            non-db
7.          website                                                 url             db
8.          webtolead_email_opt_out                                 bool            non-db
9.          webtolead_invalid_email                                 bool            non-db
```

EXEMPLO (3/5):

```
[:~$] insulin field-missing-in-bean Leads --type=url,bool,varchar --source=db --print

Found 3 missing properties in 'Leads' bean.                                    [success]

#           name                                                    type            source
1.          assistant                                               varchar         db
2.          converted                                               bool            db
3.          website                                                 url             db
```

EXEMPLO (4/5):

```
[:~$] insulin field-missing-in-bean Leads --type=url,bool,varchar --source=db --print=php

Found 3 missing properties in 'Leads' bean.                                    [success]

public $assistant;
public $converted;
public $website;
```

EXEMPLO (5/5):

```
[:~$] insulin field-missing-in-bean Leads --type=url,bool,varchar --source=db --pipe

"assistant","varchar","db"
"converted","bool","db"
"website","url","db"
```

ANEXO 7. FIELD-MISSING-IN-DATABASE

COMANDO:

```
[:~$] insulin field-missing-in-database
```

EXEMPLO (1/4):

```
[:~$] insulin help field-missing-in-database

Print information about fields declared in module(s) vardefs but not defined in respective database table(s).

Examples:
 insulin field-missing-in-database Accounts                    Print number of missing fields in Accounts database table.
 insulin field-missing-in-database Accounts --type=enum        Print number of missing fields of type enum in Accounts database
                                                               table.
 insulin field-missing-in-database Accounts --type=enum,int    Print number of missing fields of type enum or int in Accounts
                                                               module.
 insulin field-missing-in-database Accounts --print            Print list of missing fields in Accounts module.

Arguments:
 modules                                                       A comma delimited list of module names, or bean names.

Options:
 --pipe                                                        Print a compact representation of the command for scripting.
 --print                                                       Print the list of fields.
 --type                                                        A comma delimited list of field types.

Aliases: fmid
```

EXEMPLO (2/4):

```
[:~$] insulin field-missing-in-database Leads --type=date,bool --print

Found 2 missing fields in 'Leads' database table.                                                    [success]

#       name                                                  type
1.      birthdate                                             date
2.      do_not_call                                           bool
```

EXEMPLO (3/4):

```
[:~$] insulin field-missing-in-database Accounts,Leads --print

No missing fields in 'Accounts' database table.                                    [success]

Found 3 missing fields in 'Leads' database table.                                  [success]

#       name                                                    type
1.      birthdate                                               date
2.      do_not_call                                             bool
3.      website                                                 url
```

EXEMPLO (4/4):

```
[:~$] insulin field-missing-in-database Leads --pipe

"birthdate","do_not_call","website"
```

ANEXO 8. FIELD-MISSING-IN-VARDEFS

COMANDO:

```
[:~$] insulin field-missing-in-vardefs
```

EXEMPLO (1/4):

```
[:~$] insulin help field-missing-in-vardefs

Print information about fields declared in module(s) database table but not defined in respective vardefs.

Examples:
 insulin field-missing-in-vardefs Accounts                    Print number of missing fields in Accounts vardefs.
 insulin field-missing-in-vardefs Accounts --type=enum        Print number of missing fields of type enum in Accounts vardefs.
 insulin field-missing-in-vardefs Accounts --type=enum,int    Print number of missing fields of type enum or int in Accounts
                                                              vardefs.
 insulin field-missing-in-vardefs Accounts --print            Print list of missing fields in Accounts vardefs.

Arguments:
 modules                                                      A comma delimited list of module names, or bean names.

Options:
 --pipe                                                       Print a compact representation of the command for scripting.
 --print                                                      Print the list of fields.
 --type                                                       A comma delimited list of field types.

Aliases: fmiv
```

ANEXO 9. ROLE-LIST

COMANDO:

```
[:~$] insulin help role-list
```

EXEMPLO (1/2):

```
[:~$] insulin help role-list

Print information about system role(s).

Examples:
 insulin role-list --roles=1,2,Supervisors          Print informations about roles   with id or name like 1, 2 or
                                                    Supervisors.
 insulin role-list --exclude=1,2,Supervisors        Print informations about all roles except the ones with id or name
                                                    like 1, 2 or Supervisors.
 insulin role-list --exclude-tracker=no             Print informations about all roles, including the one used for
                                                    trackers.

Options:
 --roles                                            A comma delimited list of role names or role ids.
 --exclude                                          A comma delimited list of role names, or role ids, who should be
                                                    excluded.
 --exclude-tracker=<yes>                            Exclude tracker role. Choices: 'yes' (default), 'no'.
 --pipe                                             Print a compact representation of  the command for scripting.
 --print=<tabular>                                  Print the list of roles. Choices: 'tabular' (default), 'sql'.

Aliases: rlst
```

ANEXO 10. SUGAR-VERSION

COMANDO:

```
[:~$] insulin sugar-version
```

EXEMPLO (1/3):

```
[:~$] insulin help sugar-version

Print Sugar flavor, version and build number.

Options:
 --pipe                                                    Print a compact representation of the command for scripting.

Aliases: sversion
```

EXEMPLO (2/3):

```
[:~$] insulin sugar-version

Sugar CE 6.5.4 build 8477
```

EXEMPLO (3/3):

```
[:~$] insulin sugar-version --pipe

"CE","6.5.4","8477"
```

ANEXO 11. TEAM-LIST

COMANDO:

```
[:~$] insulin help team-list
```

EXEMPLO (1/2):

```
[:~$] insulin help team-list

Print information about system team(s).

Examples:
 insulin team-list --teams=1,2,"Super Heroes"          Print informations about teams with id or name like  1, 2 or "Super
                                                       Heroes"
 insulin team-list --exclude=1,2,"Super Heroes"        Print informations about all teams except the ones with id or name
                                                       like 1, 2 or "Super Heroes".
 insulin team-list --exclude-global --exclude-private  Print informations about all teams except global and private teams.

Options:
 --teams                                               A comma delimited list of team names or team ids.
 --exclude                                             A comma delimited list of team names, or team ids, who should be
                                                       excluded.
 --exclude-global=<yes>                                Exclude global team. Choices: 'yes' (default), 'no'.
 --exclude-private=<yes>                               Exclude private teams. Choices: 'yes' (default), 'no'.
 --pipe                                                Print a compact representation of  the command for scripting.
 --print=<tabular>                                     Print the list of teams. Choices: 'tabular' (default), 'sql'.

Aliases: tlst
```

ANEXO 12. USER-CREATE

COMANDO:

```
[:~$] insulin user-create
```

EXEMPLO (1/2):

```
[:~$] insulin help user-create


Create a new user account.


Examples:
 insulin user-create johndoe john.doe@example.com letmein doe     Create a new user account with the username johndoe, the email
                                                                  address john.doe@example.com, the password letmein and the last name
                                                                  doe.


Arguments:
 username                                                         The username for the new user.
 mail                                                             The email address for the new user.
 password                                                         The password for the new user.
 last-name                                                        The last name for the new user.

Options:
 --first-name                                                     The first name for the new user.
 --status=<active>                                                The status for the new user. Choices: 'active' (default), 'inactive'.
 --type=<regular>                                                 The type for the new user. Choices: 'regular' (default),
                                                                  'administrator'.


Aliases: ucrt
```

EXEMPLO (2/2):

```
[:~$] insulin user-create jmorais jmorais@mail.pt f00b4r Morais --first-name=João


 New user account created successfully.                                                                                    [success]
```

ANEXO 13. USER-INFORMATION

COMANDO:

```
[:~$] insulin user-information
```

EXEMPLO (1/4):

```
[:~$] insulin help user-information

Print information about the specified user(s).

Examples:
 insulin user-information  5,johndoe,john.doe@example.com          Print informations about user accounts with username, id or email
                                                                  5, johndoe or john.doe@example.com.

Arguments:
 users                                                            A comma delimited list of ids, usernames, or email addresses.

Options:
 --details=<profile>                                              Print user account details according to supplied detail groups. A
                                                                  comma delimited list of choices: 'profile' (default),
                                                                  'employee', 'mail', 'internal',  'layout', 'locale', 'pdf',
                                                                  'calendar' and 'all'.
 --pipe                                                           Print a compact representation of the command for scripting.

Aliases: uinf
```

EXEMPLO (2/4):

```
[:~$] insulin user-information admin

Name                            :   Super User (1)
Roles                           :
Status                          :   Active
Teams                           :
Type                            :   Administrator
Username                        :   admin
```

EXEMPLO (3/4):

```
[:~$] insulin user-information admin,jmorais@mail.pt --details=profile,mail

Name                            :   Super User (1)
Roles                           :
Status                          :   Active
Teams                           :
Type                            :   Administrator
Username                        :   admin
Mail addresses                  :   admin@mail.pt
Mail client                     :
Mail provider                   :
SMTP username                   :

Name                            :   João Morais (536453d6-9fa9-9149-fbef-503fde9918b5)
Roles                           :
Status                          :   Active
Teams                           :
Type                            :   Regular
Username                        :   jmorais
Mail addresses                  :   jmorais@mail.pt (primary), jmorais@mail.com
Mail client                     :
Mail provider                   :
SMTP username                   :
```

EXEMPLO (4/4):

```
[:~$] insulin user-information admin --pipe

"Super User (1)","","Active","","Administrator","admin"
```

ANEXO 14. VERSION

COMANDO:

```
[:~$] insulin version
```

EXEMPLO (1/3):

```
[:~$] insulin help version

Print insulin version.

Options:
 --pipe                                                    Print a compact representation of the command for scripting.
```

EXEMPLO (2/3):

```
[:~$] insulin version

insulin version 5.7
```

EXEMPLO (3/3):

```
[:~$] insulin version --pipe

5.7
```