ABAP

O guia de sobrevivência do profissional moderno

Casa do Código

MAURICIO ROBERTO CRUZ

Casa do Código
Livros para o programador
Rua Vergueiro, 3185 - 8º andar
04101-300 – Vila Mariana – São Paulo – SP – Brasil

Introdução

Em dezembro de 2010, havia acabado de sair de um emprego na fábrica de software de uma multinacional de tecnologia, e partia para desbravar águas desconhecidas. Estava trabalhando em uma consultoria nacional de SAP, alocado diretamente em clientes, e sentia-me constantemente perdido.

Após alguns meses na nova empresa, fui trabalhar em um projeto que teve um período de "baixa". Sabe como é: o cliente pede um carro importado e a consultoria projeta a implementação de uma máquina que faz o teletransporte dos carros do exterior, direto para a porta do cliente. Porém, durante a fase de construção, alguém descobre que tudo o que o cliente realmente precisava era de uma montanha de cachorros-quentes, daqueles que você compra em qualquer barraquinha de esquina.

No meio desse caos, surgem gerentes de todos os lados e criam-se milhares de reuniões de alinhamento para descobrir como transformar o teletransportador de carros em uma supermáquina de *hot-dogs*. São nessas horas que nós, os ABAPers, ficamos alguns dias parados esperando redefinições, em uma espécie de calmaria antes da tempestade. Afinal, uma supermáquina mágica de hot-dogs dessas não vai ser criada da noite para o dia. E alguém ainda vai ter de passar muita maionese no pão com uma faquinha de rocambole para agradar o cliente.

O que é que você faz quando não tem demanda? Cria um blog? É claro que não: você vai conversar no Skype com algum amigo.

Pensando no meu passado pré-ABAP, a verdade é que eu sempre tive vontade de compartilhar conhecimento, dar treinamentos, ensinar algo a alguém. Não sei explicar de que lugar isso surgiu, sei que sempre tive essa vontade. Seja na escola, em fóruns de jogos online, ou na ajuda de amigos e parentes

com vírus no computador (abraços, galera de TI), sempre tentei explicar o que eu estava fazendo para os outros de forma que eles pudessem reaproveitar o procedimento no futuro. E justamente o Mauro, o amigo com quem eu conversava no Skype, também tinha esse perfil.

Juntando duas pessoas que gostam de compartilhar conhecimento, não é de se espantar que nós tenhamos tido a ideia de compartilhar aquele monte de dicas ABAP, que guardávamos em arquivos de texto, com qualquer um que quisesse ler. Mauro mantinha um blog desatualizado de PHP no ar, que serviu como base para a idealização de um blog de ABAP. Ficamos extremamente empolgados e, em menos de um dia, tínhamos um embrião de site no ar. Só que ainda faltava o nome.

Se tem uma coisa que eu acho extremamente chata no mundo SAP é que todo mundo é extremamente formal. Parece que essa vida de trabalho em clientes consome a alma das pessoas e, aos poucos, elas tornam-se incapazes de rirem de si mesmas, transformando-se em robozinhos que se preocupam mais com formalidades do que com objetivos e prazer no trabalho.

Portanto, qual o nome que poderia quebrar essa barreira quadrada logo de cara, mostrando que ali, naquele site, poderíamos falar de SAP do nosso jeito, sem preocupações? ***ABAPZombie*** – *Prevenindo consultores de virarem zumbis*.

Assim nasceu o nome do blog. ABAPer que é ABAPer já virou noite apanhando com coisas simples e ficou parecendo um zumbi de tanto tomar café. Essa temática serviu de base para o tom dos textos e para toda a identidade visual do site. Aos poucos, ganhamos audiência, tivemos apoio da comunidade ABAP brasileira, e a motivação para continuar sempre foi crescente.

O ABAPZombie (http://www.abapzombie.com) trouxe diversas coisas boas, discussões de alto nível (algumas de baixo também), uma pancada de novos amigos e uma razão para fazer do meu próprio aprimoramento profissional uma constante. Quatro anos depois, alguém finalmente me convenceu de que eu deveria compilar parte dos artigos que escrevi em um livro, resultando nisto que você agora tem em mãos (ou na tela). Revisei todos os textos e criei muito conteúdo especialmente para esta publicação, de forma que a leitura possa ser feita de maneira contínua.

Seja você júnior, pleno ou sênior, espero que os textos consigam arrancar-

lhe algumas risadas, levantar alguns questionamentos e fazer você enxergar, ao menos, uma pequena parte do seu trabalho, de forma diferente. Prepare a sua escopeta, tire o lança-granadas do guarda-roupa e embarque comigo na jornada interminável contra os zumbis que assombram nossas vidas de ABAPers. Dica de amigo: ou você mira na cabeça, ou vai gastar munição à toa... e a nossa munição, como programadores, é o nosso tempo.

Como utilizar este livro

Para que você não fique perdido no mundo das palavras e seja mordido por um zumbi errante, os textos foram divididos em 8 capítulos e 2 apêndices:

- **Capítulo 1 – Por onde começar?**: um panorama geral da linguagem ABAP e do seu mercado, com algumas dicas de como conseguir informações para ter uma vida ABAPer menos traumática. Iniciamos também algumas discussões que serão abordadas nos próximos capítulos.
- **Capítulo 2 – A relevância do ABAP para o mercado de TI**: qual a utilização do ABAP no mundo? Quantas empresas utilizam o SAP ERP? Qual o seu passado e qual será o seu futuro? Exploramos as raízes do SAP ERP para entender alguns comportamentos do mercado, encaixando o ABAP no seu lugar da história.
- **Capítulo 3 – Certificações ABAP**: o real peso de uma certificação ABAP é constantemente questionada. Utilizando uma análise de certificações SAP, criada por alguns membros da comunidade SAP há alguns anos, compartilho minhas experiências sobre o tema.
- **Capítulo 4 – Os bastidores**: há um esforço monstruoso por trás de cada desenvolvimento ABAP que pode mudar completamente os rumos de um projeto. Entender esse processo é crucial para ter um bom futuro como ABAPer e é exatamente isso que faremos neste capítulo.
- **Capítulo 5 – Carreira, motivação, universo e tudo mais**: toda a movimentação criada nos bastidores impacta tanto a vida ABAPer, que muitos programadores só enxergam dois caminhos para evolução: a reclamação ou a desmotivação constante. Será que é possível sair desse meio e encontrar um futuro promissor?

- **Capítulo 6 – Os tabus do mundo ABAP**: a síndrome do robô atinge diversos ABAPers mundo afora, que ajudam a proliferar tabus que poderia ser resolvidos de forma bem simples. Neste capítulo, vamos refletir sobre todos os problemas clássicos da vida ABAPer: documentação, nomenclatura, comentários, uso de constantes e a satisfação do usuário.
- **Capítulo 7 – Sobre programação, sem códigos**: saber milhares de comandos e técnicas de programação é o suficiente? Há tanta coisa a ser considerada antes mesmo de uma linha de programação ABAP ser escrita que criei um capítulo só para isso.
- **Capítulo 8 – O que vem por aí**: há muitos novos produtos SAP sendo lançados e muitos deles não parecem muito "palpáveis" para os ABAPers. Vamos analisar quais são os produtos que têm mais sinergia com o desenvolvimento ABAP atual e como ficará o desenvolvimento ABAP no futuro.
- **Capítulo 9 – Apêndice I**: um guia com dicas e muito material para aqueles que quiserem aprender ABAP por conta própria, sem dependerem de um curso pago.
- **Capítulo 10 – Apêndice II**: há muitos links e citações de livros ao longo dos capítulos e esse apêndice serve como um compilado. Há também algumas indicações adicionais de conteúdo ABAP relevante.

Divirta-se!

Sobre o autor

Mauricio Roberto Cruz é bacharel em Ciência da Computação, trabalha como Arquiteto de Soluções para uma consultoria SAP, focando desde 2006 no desenvolvimento ABAP e suas ramificações. É cocriador do site ABAP-Zombie, que tenta prevenir que consultores se tornem zumbis de tanto trabalhar, compartilhando conhecimento através de artigos técnicos que ensinam de forma divertida sobre os mais diversos temas do Mundo SAP. Gosta muito de artes marciais, músicas estranhas, indie games e de testar sistemas operacionais, apps e linguagens de programação diferentes.

Lista de discussão

Caso você queira conversar comigo e com outros leitores sobre alguns dos tópicos ou assuntos deste livro, inscreva-se na lista de discussão (via e-mail), por meio do link: http://forum.casadocodigo.com.br/.

Agradecimentos

Para todos aqueles que acessam o ABAPZombie

A criação deste livro não existiria sem as pessoas que acessam o ABAPZombie diariamente. Agradeço imensamente a todos pelos feedbacks, críticas, discussões e risadas compartilhadas por meio do site. Muito obrigado a todos os consultores que deixaram de virar zumbis por conta nossos textos!

Para a galera do SELECT em LOOP

Muita gente acha que sou maluco, mas maluco que é maluco precisa compartilhar sua maluquice com alguém. Sem o suporte diário da minha esposa, Stella, meus projetos não tomariam vida. Ela é quem me mantém no caminho, com conselhos que potencializam positivamente tudo o que faço. Palavras nunca serão suficientes para expressar a minha gratidão.

Véio, sem o suporte do mano Mauro esse rolê não teria visto a luz do dia. Esse malandro é cheio de fitas, quebradas, gols quadrados, ZLs, motocas, vermes e uma amizade sem limites. Valeu maaano! Juntos continuaremos nossa disputa diária contra a nossa arqui-inimiga, uma tal de Priscila, a quem também agradeço pelas conversas, risadas, aranhas e doideiras. Valeu maaaano ao quadrado!

Aos outros 3 *brothers* que também mantêm a máquina ABAPZombie funcionando: mano do Metal, mano Mentorado e mana da Tattoo – obrigado por fazerem parte da equipe!

O Danilo precisou falar comigo umas 4 vezes até eu aceitar a ideia do livro. Obrigado por não ter desistido de me convencer dessa ideia e por ser um dos meus principais comparsas na busca pelo entendimento deste mundo.

Ronildo, Rabay e Danilo (de novo), obrigado por terem me ajudado na revisão de conteúdo do livro. O feedback de vocês foi extremamente im-

portante, e eu precisarei pagar cafés a vocês até o infinito para compensar! Agradeço também à Casa do Código pela oportunidade, especialmente para a Vivian por ter me ajudado tanto durante a escrita. Valeu!

Eu trabalhei com tantas pessoas em clientes, projetos e consultorias, que seria impossível de citar todo mundo. Saibam que todos contêm uma parcela de responsabilidade por tudo que cerca este livro. Meu muito obrigado aos ABAPers, funcionais, BASIS, gerentes, sócios, líderes e `<insiraumcargo>` com quem trabalhei.

Por fim, deixo os meus agradecimentos para toda a galera da comunidade SAP que conheci por conta do ABAPZombie. Muito obrigado por apoiarem o meu trabalho, e compartilharem momentos polêmicos e cômicos do dia a dia!

Para aqueles que pensam que eu sei arrumar impressoras

Dionisio, Luiza e Marcos. Três pessoas que não contêm a mesma paixão que eu tenho por tecnologia, mas sempre me apoiam cegamente em minhas escolhas. Dedicarei o meu melhor para sempre trazer felicidade a vocês através desse monte de nomes, siglas e aparelhos "estranhos". Muito obrigado, família.

Em diversos momentos da minha vida, deparo-me com momentos nostálgicos por conta das aventuras *undergrounds* que tive com meus amigos Otit, Red Arsenal e Carzombie. Nossa amizade e história são muito importantes. Obrigado!

O envolvimento com artes marciais mudou (e continua mudando) a minha vida. Agradecerei eternamente ao meu mestre e amigo Celso Rodrigues, por tudo que aprendi nos anos em que treinei Hapkido. Muito obrigado! Agradeço também meus Senseis de Kendo, Yoshiaki Kishikawa e Mitiko Kishikawa, que desenvolvem um trabalho fantástico, e contribuem para que eu mantenha minha mente serena e meu espírito forte.

Abraços a todos aqueles que estão lendo, leram ou lerão este livro! :)

Sumário

Capítulo 1

Por onde começar?

É quase sempre a mesma história: algum profissional SAP dá ao seu amigo a famosa dica de que o mercado SAP é a última maravilha do milênio, com uma demanda altíssima de profissionais e salários bem acima da média brasileira. Não é difícil encontrar histórias de profissionais sem formação em TI que largaram tudo para tentar uma vaga no mundo SAP, mas é realmente custoso encontrar pessoas que fizeram isso e tiveram sucesso na empreitada.

Com algumas dicas simples, é possível prevenir-se de um início traumático. A análise prévia de como o mercado comporta-se e quais são as suas opções poderá salvar uma grande quantidade do seu tempo e dinheiro. Neste capítulo, vamos começar a entender um pouco sobre o que é ABAP, como o mercado funciona e quais as opções para que você possa ingressar nessa área.

1.1 Quem é SAP e o que é o ABAP?

SAP é uma empresa alemã especializada em *enterprise software* (software corporativo). Ela possui um portfólio enorme de produtos para ajudar empresas a maximizarem os seus negócios. Seu maior produto ainda é o SAP ERP, um grande pacote de ferramentas que permite a criação de programas para praticamente qualquer área da empresa. Com ele, é possível integrar diversas áreas e processos em um só lugar.

O produto é um gigante do mundo empresarial e é amplamente usado por empresas de grande porte, como veremos no capítulo 2. Entretanto, é importante destacar que, nos últimos anos, a empresa vem buscando a diversificação de produtos, e áreas de atuação com novidades que envolvam o trabalho de vários tipos de desenvolvedores e profissionais de TI, conforme apresentaremos no capítulo 8.

Toda a programação do SAP ERP é feita utilizando o ABAP (*Advanced Business Application Platform*), linguagem de programação de 4ª geração, criada nos anos 80 junto ao SAP ERP. O ABAP é moldado especialmente para lidar com as diferentes partes do ERP.

Ele contém uma série de funcionalidades para facilitar a implantação de regras de negócio complexas e a manipulação de uma enorme quantidade de dados. Ele também é capaz de fazer integração de sistemas externos com o SAP e criar interfaces (telas) para os usuários, acessíveis por meio de transações do ERP (uma espécie de link para um programa).

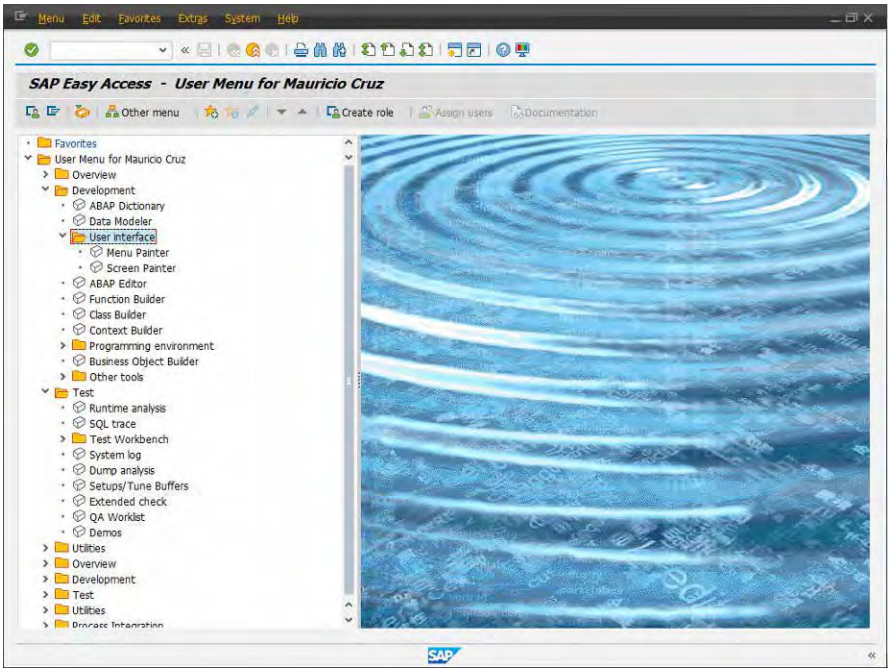


Fig. 1.1: Tela inicial do SAP ERP feita em ABAP

Quando um cliente implementa o SAP ERP em sua empresa, ele ganha, por padrão, uma série de programas ABAP criados pela própria SAP. Esses programas contêm lógicas prontas para resolver diversos problemas comuns nas empresas, além de integrarem o funcionamento de áreas distintas. São popularmente conhecidos como **programas standard**, criados e mantidos pela própria SAP. Ao longo do tempo, é comum que novos programas standards sejam feitos e que os antigos sejam atualizados.

Os desenvolvedores ABAP (ou ABAPers) que não estão ligados com a SAP vão até as empresas para implantar regras de negócio específicas. Cada empresa tem alguma particularidade em seu processo, que não tem relação com outras empresas do seu segmento. Essas lógicas são criadas dentro do ERP, em **programas customizados**. Quem cria e especifica esses programas são os funcionais, profissionais capacitados em entender o processo do cliente e estruturar o fluxo da informação dentro do SAP. O ABAPer criará os

programas com base nas regras descritas nessas especificações.

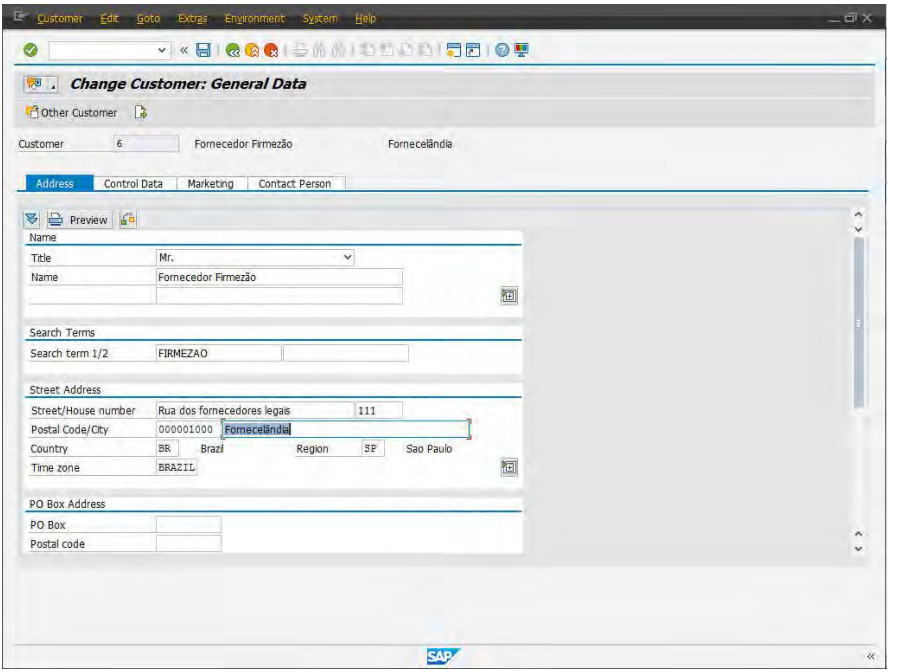


Fig. 1.2: Exemplo simples de uma tela criada com o ABAP clássico

O foco deste livro é o ABAP, mas é importante saber que o trabalho do ABAPer requer uma série de interações com diferentes tipos de profissionais que trabalham na área. Usar a expressão *mundo SAP* não é nenhum exagero, dada a quantidade de pessoas com as mais diferentes formações e habilidades (RH, financeiro, administração, vendas, materiais e mais 20 áreas corporativas à sua escolha). Mesmo dentro da área de programação ABAP, existem diversas ramificações e especializações em ferramentas, como: Webdynpro ABAP, Workflow, CRM, SRM, HR e APO, todos produtos e extensões da SAP que requerem programação ABAP em sua implantação.

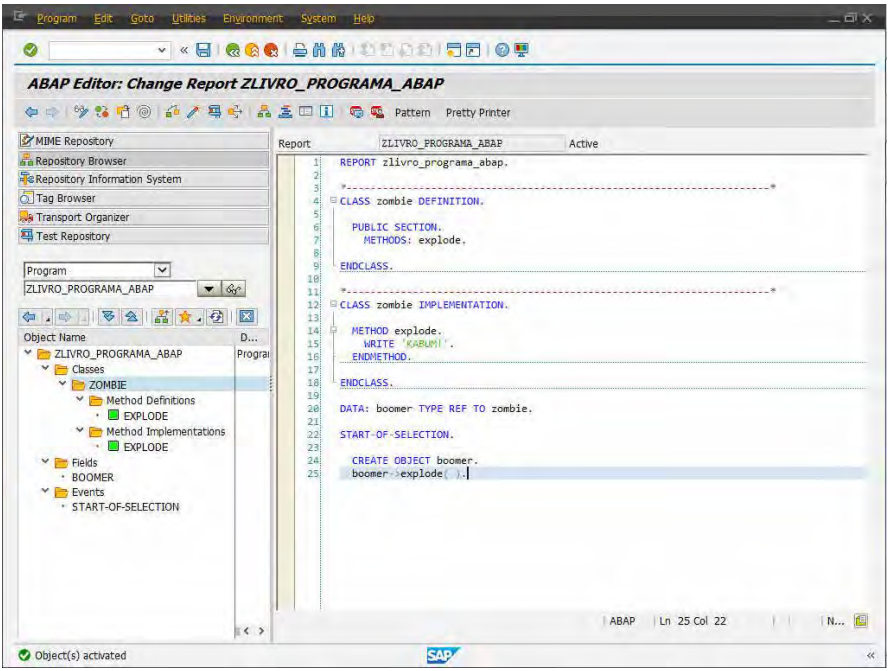


Fig. 1.3: Editor ABAP acessado por meio da SAPGui: um cliente para exibição das telas do SAP ERP

Há um fato que pode parecer estranho para quem não é iniciado na arte SAP: as empresas que possuem o SAP ERP normalmente não contam com grandes times internos para fazer as novas customizações. Geralmente, elas contam com times que ajudam a manter o sistema funcionando (uma equipe de suporte), e preferem contratar profissionais externos, de tempos em tempos, para implantar novas funcionalidades (ou automatizações) no formato de programas ABAP, organizando essas atividades em um projeto.

Se elas precisarem desses profissionais SAP "externos", a quem elas podem recorrer?

1.2 Consultorias SAP

As empresas que contêm a mão de obra especializada do mundo SAP são as consultorias. Elas servem como um intermediário entre a necessidade do cliente e os profissionais SAP. Em resumo, elas são especializadas em terceirizar o trabalho, montando equipes e/ou encontrando profissionais para suprir as demandas dos clientes.

Trilhando este caminho, há grandes chances de você ingressar no mundo SAP, trabalhando para uma consultoria, pois o número de ABAPers trabalhando diretamente em clientes não costuma ser muito grande. Também não é comum que empresas terceirizem o trabalho ABAP sem o intermédio das consultorias, fazendo um contrato direto com os profissionais – há lendas urbanas de isso ter acontecido muito no passado, mas, hoje em dia, há uma série de questões políticas/trabalhistas nas empresas que tornam contratos desse tipo extremamente raros.

Existem consultorias dos mais diversos tipos, das gigantes e multinacionais até as pequenas que não possuem nem um escritório, atuando somente como intermediárias virtuais. Elas são motivos de amor e ódio, fruto de um mercado que contém uma concorrência acirrada. É comum escutar histórias de consultorias "mercenárias", por não tratarem bem seus profissionais, como também encontrar ABAPers que trabalham há muitos anos para a mesma consultoria, por manterem uma relação interna saudável e livre de complicações.

A grande questão para os iniciantes é como conseguir um emprego nessas consultorias. Os clientes dificilmente pagarão por profissionais que estão aprendendo, os juniores. As demandas geralmente exigem níveis pleno/sênior, e isso cria uma barreira imensa de entrada. Mesmo com essa escassez de trabalho para os iniciantes, as consultorias sabem que é preciso formar novos profissionais para o mercado; caso contrário, elas ficarão eternamente degladiando-se pelas mesmas pessoas.

Na sequência, vamos analisar algumas formas de ingressar na área.

1.3 Academias seletivas

É muito popular o uso do termo "academia" para designar treinamentos do mundo SAP. Elas costumam ser cursos extensos, atingindo um nível considerável de detalhes. Nem todas as áreas do mundo SAP dispõem de ambientes gratuitos para teste e estudo (como as áreas "funcionais"), o que reforça o formato longo do curso.

Há muitas academias pagas ofertadas no mercado brasileiro e mundial – presenciais ou online –, cobrindo praticamente qualquer assunto relacionado aos produtos SAP. A academia seletiva é o modo que considero mais cômodo para iniciantes. Acabei entrando nesse mundo maluco dos ERPs por ela.

> É importante que eu lhe avise que "academia seletiva" é um nome inventado por mim, para explicar essa prática. Não vá bater na porta de empresas falando disso (mas, se fizer, mande-me um e-mail contando como foi).

Ela funciona assim: a empresa usa a academia de ensino da linguagem e o ecossistema ABAP como parte do seu processo seletivo. Você tem a chance de aprender a trabalhar com ABAP sem custo, mas a empresa reserva o direito de não lhe contratar, caso você não corresponda às expectativas.

Essas vagas normalmente são abertas para estágios, tendo como alvo principal estudantes que estão cursando a faculdade, ou que são recém-formados. Existem casos de esta modalidade ter tido como alvo pessoas que estão migrando para o SAP, vindo de outras tecnologias, mas são casos bem menos frequentes.

Geralmente, empresas consideradas grandes no mundo das consultorias de TI têm essa postura, como a primeira empresa para qual trabalhei como ABAPer. No meu caso, posso dizer que não fui eu que escolhi o ABAP, na verdade foi algum profissional da área de R&S (Recrutamento e Seleção).

Quando eu estava fazendo o processo seletivo, existiam duas academias-seletivas: uma para Java, outra para ABAP. Seja pelo alinhamento dos planetas, pelo humor do recrutador, ou pela minha cara de SAP(o?), fui parar na academia ABAP. E é por isso que hoje não existe um site chamado Javazombie.

Se você estiver na faculdade – maior público alvo de empresas com essa modalidade de recrutamento –, esse é, de longe, o melhor caminho. Prepare-se para um processo seletivo complexo (comum nesse tipo de vaga), mas que será extremamente gratificante se concluído com sucesso. Obviamente, a consultoria esperará que você mantenha uma relação de longo prazo com ela, de forma que ela possa formar um profissional a seus moldes.

1.4 Cursos de ABAP

Várias empresas ministram cursos pagos de ABAP, e de várias outras áreas de SAP, em diversas cidades do Brasil. Você pode fazer o curso na própria SAP, em uma escola autorizada que utiliza o material da SAP, ou em uma escola que ministra o curso usando material próprio.

Entretanto, a realidade do dia a dia nos mostra que ter feito um desses cursos de ABAP não garantirá que você consiga uma posição no mercado em curto prazo (às vezes, nem mesmo a longo prazo). Muita gente acha que é fácil conseguir emprego no mundo SAP, por conta da quantidade enorme de vagas não preenchidas. Entretanto, você já se perguntou quantas dessas vagas são para consultores juniores?

Muitas consultorias de pequeno e médio porte (maiores ofertadoras de vagas) têm dificuldade em manter profissionais novos, já que as empresas que têm o SAP ERP instalado sempre dão preferência aos profissionais experientes para desenvolver seus projetos. Como a consultoria vive da prestação de serviços, o seu foco principal é manter uma equipe com um nível técnico alto, para prover serviços de qualidade, o que gera uma briga pelos mesmos profissionais.

Nessa história toda, quem sofre são os iniciantes na área. É muito comum encontrar relatos de pessoas que fizeram o curso, gastando uma quantidade considerável de tempo e dinheiro, para falhar na hora de conseguir uma posição de júnior.

É claro que o curso é um ponto de partida, mas você precisa ser realista quando avaliar as opções. Não vá achando que você vai concluir o curso e conseguirá um emprego na próxima semana. Isso só acontecerá se você encontrar consultorias com demanda para juniores – o que é raro –, ou se tiver

um QI (quem-indica) de alto nível.

Há quem diga que um certificado pode ser um grande diferencial, mas quem é da área sabe que o tema certificação está relacionado a uma série de polêmicas, como vamos explorar no capítulo 3.

Mas acalme-se, não estou dizendo que é impossível virar programador ABAP. Pelo contrário! É importante você saber que não vai ser tão fácil como aquele seu amigo que já está na área diz, e que você terá de se dedicar muito para conquistar o seu espaço. O curso é só o primeiro passo de uma grande caminhada.

E se você fizesse tudo por conta própria?

Os passos descritos neste capítulo dão uma ideia geral de como o mundo ABAP funciona. Muita gente segue o caminho natural: fazer uma academia, pagando por um curso, para aprender a trabalhar na linguagem. Mas há uma outra forma: aprender tudo sozinho! Se você é do tipo que gosta de estudar por conta própria e não desiste frente à primeira dificuldade, aproveite as dicas do apêndice, no capítulo 9, para ingressar direto no mundo ABAP, enquanto lê o resto do livro. :)

Importante: se você quer virar um programador ABAP, experiências prévias com programação vão ajudá-lo a se destacar. No mundo do desenvolvimento de software, não é a linguagem de programação que vai dizer se um programador é realmente bom ou não. Um bom programador deveria conseguir transitar tranquilamente entre mais de uma linguagem.

E por falar de profissionais de outras áreas...

1.5 Trabalho com XYZ e quero migrar para SAP

O mundo SAP atrai todo tipo de gente (já vi advogados e até dentistas), normalmente por conta dos salários: a pessoa encontra alguma pesquisa salarial de SAP na internet (entendeu a dica, certo?) e vê que um emprego similar ao seu, no mundo SAP, tem um salário mais alto. A principal dúvida é se a sua bagagem e experiência de outros sistemas/linguagens lhe dará alguma vantagem na hora da procura do novo emprego.

Eu, sinceramente (ou será infelizmente?), acho que depende muito.

Muitas consultorias trabalham com alocações por projetos, e elas são as que mais ofertam vagas no mercado. Esse tipo de consultoria dificilmente levará em conta alguma bagagem prévia no momento da contratação: se você acabou de fazer a academia ABAP e é um júnior, não vai importar muito se você trabalhou 10 anos com COBOL (*COmmon Business Oriented Language*).

"*Você é um júnior. Não tem experiência e não terá espaço em um projeto onde tudo precisa ser feito em prazos apertados*", diria a consultoria. Pode parecer uma prática mercenária, mas é um meio que algumas consultorias encontram para sobreviver no mercado, já que enviar um profissional que não resolva o problema do cliente pode resultar no demérito do trabalho da consultoria como um todo. Se isso é certo ou errado é uma discussão à parte.

Porém, existem algumas consultorias que mantêm profissionais SAP como parte de sua equipe – em vez de alocar por projetos –, fazendo com que eles transitem em seus clientes. Esse tipo de consultoria deve ser o seu foco principal, pois é interessante para ela manter uma equipe diversificada e que possa atender clientes em momentos-chave.

Nesse caso, a sua experiência de 5 anos trabalhando com banco de dados pode vir a calhar, quando surgir uma integração com algum banco de dados externo. A SAP também é conhecida por lançar uma série de produtos novos a cada ano, e muitos deles transitam em outras áreas, como Android, iOS, HTML5, CSS3, Java etc.

Se você pretende mesmo mudar para SAP, invista fortemente e demonstre que a sua experiência fará a diferença para seu aprendizado, encurtando o tempo até atingir um nível em que você consiga "andar com as próprias pernas". Só não se assuste caso alguém ignore sua experiência prévia, mesmo sendo algo relacionado à TI. Sinceramente (ou infelizmente?), a chance de isso acontecer é bem alta.

Mas o ABAP é muito antiquado...

É muito comum escutar esse tipo de frase. O ABAP é realmente uma linguagem de programação antiga, mas está em constante evolução.

Novas *features* são criadas todos os anos, e muitas delas são equivalentes a features de linguagens de programação modernas. O que gera muito desses

comentários é o fato de a SAP manter a retrocompatibilidade em seu sistema, ou seja, uma determinada funcionalidade ABAP, criada nos anos 80, pode continuar funcionando até os dias de hoje.

Um ABAPer que teve uma formação nos anos 90 e não se atualizou **ainda** consegue entregar programas ABAP que funcionam para os usuários. Esse fenômeno gera uma série de problemas na área e será bastante abordado no decorrer do livro.

O principal é saber que existe inovação e evolução no mundo SAP, porém a implantação dessas inovações dependerá sempre dos clientes. Já que os ABAPers prestam serviços para as empresas, eles sempre dependerão delas implantarem novos produtos da SAP, para utilizar as novas funções da linguagem ABAP – é possível testar em ambientes *Trial*, mas a implantação real depende das empresas.

A dinâmica é muito diferente de outros mercados de TI, como o de desenvolvimento Web, onde a chance de você usar algo novo é uma constante e depende mais das diretrizes definidas pela equipe do seu projeto, do que de algum contrato multimilionário.

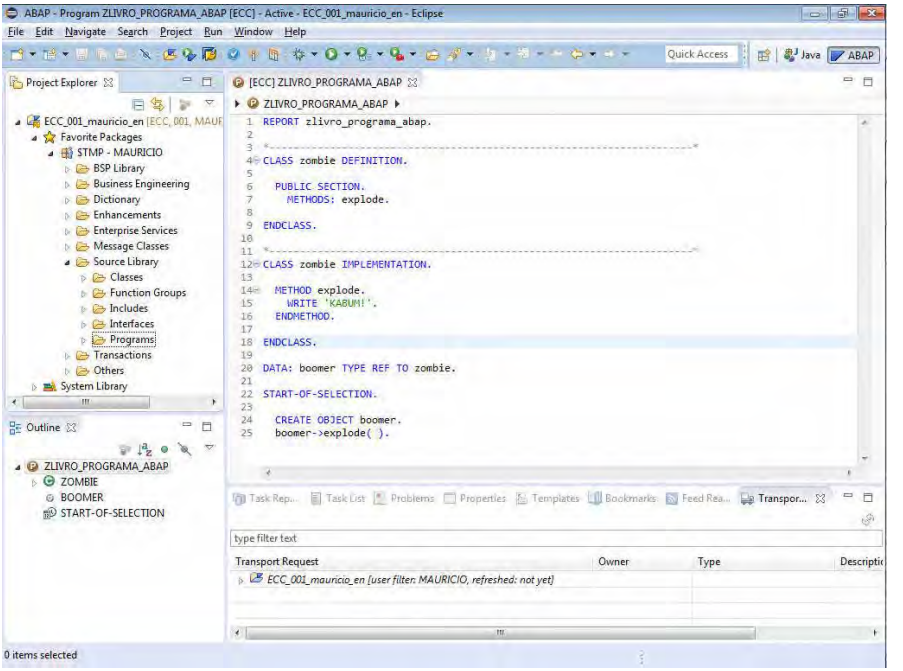


Fig. 1.4: Um dos exemplos dessa evolução é que, há alguns anos, já é possível programar em ABAP pelo Eclipse

De qualquer forma, a disposição para enfrentar esse mercado, trazendo ideias inovadoras, pode ser um grande diferencial. Talvez você precise assimilar algumas coisas do passado primeiro, mas existem muitos ABAPers interessados em aprender, implantar e desenvolver essas novas tecnologias.

1.6 A BUSCA POR VAGAS

Com todo esse entendimento das opções, é importante saber onde analisar as vagas que são ofertadas diariamente no mercado. Apesar da popularização do LinkedIn, boa parte das vagas do mundo SAP do Brasil ainda é anunciada por listas de e-mail, que nasceram inicialmente como listas de discussão, e acabaram virando listas de vagas.

Vez ou outra você encontrará algumas discussões sobre coisas técnicas,

mas aposto que você vai receber muito mais e-mails com vagas. São tantos por dia que acho válido você criar uma conta separada, só para assinar essas listas, a não ser que você não tenha problemas em lidar com uma caixa de entrada caótica.

Listas que você pode seguir para vagas ABAP:

- abap-sap@grupos.com.br
- abap-4@yahoogrupos.com.br

Se você não sabe como entrar em um grupo de e-mail, pesquise!

Quase todo mundo de SAP ABAP no Brasil segue essas listas, portanto, preciso ressaltar uma coisa que vejo toda semana: se você for mandar seu CV para alguma vaga, cuidado para não enviá-lo para o grupo todo. Parece uma coisa básica, mas toda semana alguém comete esse erro. Veja no e-mail da vaga para qual endereço de e-mail você tem de mandar seu CV, e mande somente para ele. Obrigado. :)

O Facebook também está popularizando-se no compartilhamento de vagas. Procure pelos grupos de SAP e ABAP, e fique ligado nas vagas que as recrutadoras compartilham.

Taxa Hora e a Arte de ser um "Sombra"

Você fez o curso, entrou nas listas, viu milhões de vagas... E sempre a pessoa que está recrutando pede a "pretensão salarial".

Se você é um iniciante, é óbvio que você não vai saber quanto pedir. Como já comentei, existem pesquisas salariais públicas de SAP na internet, para que você tenha uma base dos valores. Se for a sua primeira oportunidade, não exija valores altos, principalmente porque você estará concorrendo com outros iniciantes. Neste momento, dê prioridade a adquirir conhecimento. Preocupe-se com salários após ter condições técnicas de exigir um valor adequado.

Diz a lenda que algumas consultorias contratam juniores para trabalharem em projetos como *Sombras*. Um Sombra é uma pessoa que não tem custo para o projeto e está ali unicamente para ficar ao lado de alguém mais experiente, a fim de aprender o trabalho e, posteriormente, ser contratado como júnior.

Apesar de eu ter dito "lenda", já ouvi relatos de que esse tipo de contratação existe, porém é bem rara. Se você encontrar algo do tipo e não tiver uma necessidade monetária urgente, agarre essa oportunidade. Aprender na prática é sempre melhor.

1.7 Eu já fiz o curso e tal... Posso parar de estudar?

Claro que não! O trabalho na área de tecnologia muitas vezes requer que você encontre respostas para questões do dia a dia por conta própria. Você vai utilizar muito o Google, pode apostar.

Há também enormes chances de você encontrar respostas no site que a própria SAP mantém para sua comunidade de profissionais, chamado *SAP Community Network* (SCN). Você encontrará toneladas de informações sobre (quase) qualquer coisa de que você precise, como: apostilas, *e-learning*, fóruns de discussão, blogs, apresentações, wikis etc.

O SCN é o principal portal para técnicos do mundo SAP. A leitura recorrente dos artigos e discussões são essenciais para evoluir na carreira como ABAPer.

Uma outra ótima fonte para estudos sem custos é o site OpenSAP (http://open.sap.com/) , que contém cursos completos, feitos por meio de vídeos, com o material preparado pela própria SAP. E o melhor: é tudo completamente gratuito. Mas adivinhe só: é tudo em inglês.

Fica aqui a **principal** dica deste livro: se você não sabe inglês e quer trabalhar com SAP (ou TI), saiba que você não vai poder aproveitar quase nada do que tem na internet e vai perder muitas informações valiosas que certamente iriam ajudá-lo no dia a dia. Existem blogs nacionais como o ABAPZombie e o ABAP101, mas a quantidade de informação é muito pequena em comparação ao volume monstruoso gerado pela comunidade internacional.

Comunidades de outras linguagens têm fóruns nacionais com muitas informações, como o GUJ (Java), mas ainda não temos nada similar que funcione tão bem para ABAP, e acho pouco provável que isso aconteça no futuro.

Virando o ABAP-Master-Blaster

Seguindo essas dicas, é possível virar o SAP-Mega-Master-Blaster? Depende!

Depende do quê? De sua dedicação e de sua vontade de ter sucesso nessa área.

Eu defendo que todo mundo é capaz de fazer qualquer coisa, desde que tenha vontade e dedicação. O caminho pode ser árduo, mas, com paciência e persistência, você chega lá. Mais uma vez: conseguir um emprego como júnior é só o primeiro passo. Um complicado passo para alguns, é verdade, mas somente o primeiro de uma jornada recheada de desafios.

Leia relatos de iniciantes

Escrevi pela primeira vez sobre iniciantes em um post do ABAPZombie, chamado de *Guia do ABAP Noob* (http://bit.ly/2biqp6f) . A versão deste livro é mais completa e atualizada, mas esse post contém diversos comentários, com muitos relatos interessantes sobre os passos iniciais do mundo SAP.

Recomendo a leitura dos textos para todos da área!

Capítulo 2

A relevância do ABAP para o mercado de TI

O ABAP não é uma linguagem de programação popular. Há grandes chances de programadores mais novos não fazerem a menor ideia de que ele sequer existe. Eu mesmo, quando comecei no mundo SAP e ainda estava cursando a faculdade, tive de explicar para mais de um dos meus professores do que se tratava o tal ABAP com que eu trabalhava. É claro que, com meus sólidos conhecimentos de estagiário de 2 meses, devo ter dado uma explicação completamente superficial e contribuído com a impopularidade da coisa toda.

No começo, eu procurava explicar o ABAP relacionando-o com a utilidade de um ERP para uma empresa. E as pessoas me olhavam estranho. Mas depois de um tempo, enfim, cheguei a uma explicação bem mais compreensível para humanos que entendem (ou não) códigos binários: se você consome

produtos ou serviços de grandes corporações, há grandes chances de o ABAP fazer parte da sua vida, sem que você sequer faça ideia. É para criar programas para otimizar processos dessas grandes organizações que o ABAPer existe.

Para que você tenha uma dimensão do tamanho do ecossistema, dados oficiais divulgados pela SAP (em janeiro de 2015) indicam que 98% das 100 empresas mais valiosas do mundo contêm algum tipo de produto SAP em suas operações. Das 2.000 empresas listadas como as maiores empresas públicas pela Forbes, 87% dela são clientes SAP.

Os valores movimentados anualmente nesse mercado ultrapassam facilmente a casa dos bilhões (de euros). Daria para comprar toda a máquina que movimenta a MegaSena e ainda sobraria (muito) dinheiro.

Se esse mercado é tão gigantesco e impacta a vida de tantas pessoas (mesmo que por tabela), por que ele é tão desconhecido, mesmo entre os envolvidos com o mercado de TI? A resposta dessa pergunta está diretamente relacionada com as raízes da SAP como empresa. Neste capítulo, vamos avaliar brevemente a história da SAP para entender por que este é um mundo tão "fechado".

2.1 Os primórdios do ABAP

A SAP é uma empresa anciã, se considerarmos a dinâmica da criação das empresas dos tempos atuais. Foi criada no início dos anos 70, por um grupo de cinco ex-funcionários da IBM, com o foco de automatizar e integrar processos corporativos através de softwares.

Já naquela época, a necessidade de ter dados relevantes em tempo real era algo muito importante para empresas. E o sistema da SAP focava exatamente nisso, gerando informações quando elas eram requeridas em vez de fazê-lo durante processamentos noturnos (era muito comum rodar processamentos complexos de informações à noite, quando ninguém estava trabalhando).

Esse sistema integrado e com processamento em tempo real evoluiu até chegar no chamado sistema R/2, que teve sua primeira versão lançada no final dos anos 70. Foi o produto principal da SAP durante todo a década de 80. Nessa época, ABAP já dava as caras, engatinhando como uma linguagem para gerar relatórios.

Mas foi somente em 1992, com a criação do chamado SAP R/3 (ERP com arquitetura em 3 camadas – apresentação, aplicação e banco de dados), que o ABAP ganhou força, como base de programação de todo sistema. Praticamente todas as aplicações e processos internos do R/3 foram criadas em ABAP, sendo também a linguagem utilizada para criar extensões no sistema, com base nos requerimentos de cada empresa.

O sistema é separado em módulos (vendas e distribuição, gerenciamento de materiais, RH etc.) e é muito flexível, podendo ser implantado em empresas de segmentos completamente distintos (por exemplo, empresas aeronáuticas e de bens de consumo).

O ABAP nasceu como uma linguagem com comando extensos. Para fazer uma simples leitura de um dado em uma tabela interna (uma espécie de array das outras linguagens), era necessária uma série de palavras: `READ TABLE tabela INTO linha WITH KEY <igualdade> INDEX 1.`. Extremamente trabalhoso (ou chato).

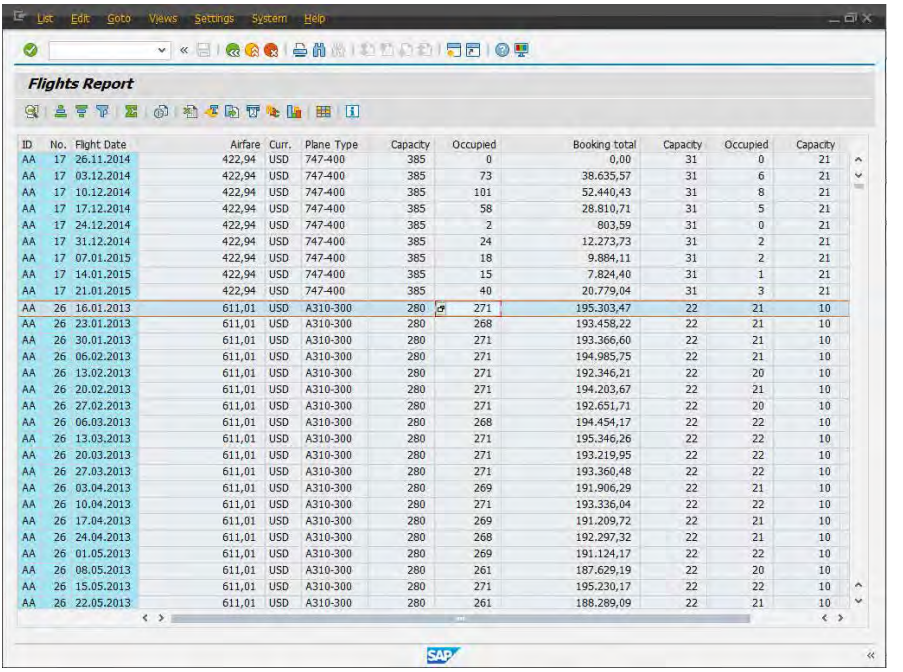
Por este motivo, é naturalmente comparada ao COBOL, que possui sintaxe semelhante. A linguagem recebeu muitas atualizações desde a sua criação, ao ponto que o mesmo comando pode ser escrito de forma similar a linguagens modernas – `tabela[ 1 ]` –, desde que o sistema do cliente esteja atualizado com os últimos *patches*.

Apesar das atualizações constantes, ainda é possível utilizar o mesmo bom e "velho" ABAP dos anos 90 nos dias atuais, por conta da retrocompatibilidade mantida pelo sistema. Vamos explorar muito como essa retrocompatibilidade afetam os ABAPers e práticas de hoje em dia, no capítulo 6.

O R/3 foi o sistema ERP que fez com a SAP iniciasse uma penetração de mercado tremenda, que se sustenta até os dias atuais. O ABAP é responsável por grande parte dessa popularidade, por ser uma linguagem fácil de ser aprendida e muito produtiva.

Um relatório de materiais com diversas funcionalidades (ordenação, layouts diferentes, filtros, seleção de linhas/colunas, entre outras) pode ser criado por qualquer ABAPer mediano em minutos, seja hoje ou no início dos anos 90. Também não é necessário entender de particularidades do banco, pois o ABAP abstrai completamente os acessos por meio do Open SQL, um conjunto de comandos ABAP que abstrai as operações, funcionando em qual-

quer banco de dados suportado pela SAP (praticamente todos os bancos usados por grandes empresas são suportados).



ID	No.	Flight Date	Airfare	Curr.	Plane Type	Capacity	Occupied	Booking total	Capacity	Occupied	Capacity
AA	17	26.11.2014	422,94	USD	747-400	385	0	0,00	31	0	21
AA	17	03.12.2014	422,94	USD	747-400	385	73	38.635,57	31	6	21
AA	17	10.12.2014	422,94	USD	747-400	385	101	52.440,43	31	8	21
AA	17	17.12.2014	422,94	USD	747-400	385	58	28.810,71	31	5	21
AA	17	24.12.2014	422,94	USD	747-400	385	2	803,59	31	0	21
AA	17	31.12.2014	422,94	USD	747-400	385	24	12.273,73	31	2	21
AA	17	07.01.2015	422,94	USD	747-400	385	18	9.884,11	31	2	21
AA	17	14.01.2015	422,94	USD	747-400	385	15	7.824,40	31	1	21
AA	17	21.01.2015	422,94	USD	747-400	385	40	20.779,04	31	3	21
AA	26	16.01.2013	611,01	USD	A310-300	280	271	195.303,47	22	21	10
AA	26	23.01.2013	611,01	USD	A310-300	280	268	193.458,22	22	21	10
AA	26	30.01.2013	611,01	USD	A310-300	280	271	193.366,60	22	21	10
AA	26	06.02.2013	611,01	USD	A310-300	280	271	194.985,75	22	21	10
AA	26	13.02.2013	611,01	USD	A310-300	280	271	192.346,21	22	20	10
AA	26	20.02.2013	611,01	USD	A310-300	280	271	194.203,67	22	21	10
AA	26	27.02.2013	611,01	USD	A310-300	280	271	192.651,71	22	20	10
AA	26	06.03.2013	611,01	USD	A310-300	280	268	194.454,17	22	22	10
AA	26	13.03.2013	611,01	USD	A310-300	280	271	195.346,26	22	22	10
AA	26	20.03.2013	611,01	USD	A310-300	280	271	193.219,95	22	22	10
AA	26	27.03.2013	611,01	USD	A310-300	280	271	193.360,48	22	22	10
AA	26	03.04.2013	611,01	USD	A310-300	280	269	191.906,29	22	21	10
AA	26	10.04.2013	611,01	USD	A310-300	280	271	193.336,04	22	22	10
AA	26	17.04.2013	611,01	USD	A310-300	280	269	191.209,72	22	21	10
AA	26	24.04.2013	611,01	USD	A310-300	280	268	192.297,32	22	21	10
AA	26	01.05.2013	611,01	USD	A310-300	280	269	191.124,17	22	22	10
AA	26	08.05.2013	611,01	USD	A310-300	280	261	187.629,19	22	20	10
AA	26	15.05.2013	611,01	USD	A310-300	280	271	195.230,17	22	22	10
AA	26	22.05.2013	611,01	USD	A310-300	280	261	188.289,09	22	21	10

Fig. 2.1: Com os dados montados, um relatório como esse fica pronto em poucos minutos

Além do ABAP, uma série de complementos e outros produtos também foram gerados pela SAP, desde a criação do R/3, que compõe o ecossistema SAP. São os mais diversos aplicativos: alguns focados na integração de sistemas (PI), outros com foco no mundo mobile (SMP) e até mesmo ERPs menores para pequenas e médias empresas (SAP Business One).

Há também um pacote de produtos criados em cima do chamado SAP Web Application Server (WebAS) e uma plataforma criada primariamente em ABAP, que serve de berço para diversos produtos da SAP (CRM, SCM, o próprio ERP etc.).

Muitas siglas e nomes confusos, não é mesmo? O número de produtos e soluções gerados desde a ascensão do R/3 culminam em um cenário extrema-

mente complexo, tanto que, às vezes, nem mesmo o cliente sabe aquilo que contratou.

Já vi alguns pagarem horas de consultoria para entender o que fazem produtos adicionais embutidos em licenças que eles mesmos possuem. O mundo corporativo pode ser tão confuso que o cliente pode acabar pagando uma licença para algo que ele não faz a mínima ideia se é útil para seu negócio. Parece loucura, já que os contratos envolvem montantes altos, mas é uma situação comum. Portanto, se você ficou um pouco perdido, fique tranquilo que você não está sozinho (e pelo menos você não perdeu milhões com isso).

Avaliando o império criado pela SAP ao longo dos anos, fica claro que o ABAP – como linguagem – e a SAP – como provedora de softwares – são extremamente relevantes para o mercado de TI. Mas é importante notar que é um mercado focado **somente** em empresas, e não ao usuário final.

Isso implica em diversas diferenças com mercados mais abertos, como o mercado web, de games ou mobile, sendo a principal delas essa característica de ser um mercado pouco acessível, a não ser que você tenha envolvimento direto com as empresas que utilizam os produtos SAP.

2.2 Sem SAP, sem ABAP

A grande questão é que sem um sistema SAP, simplesmente não existe programação ABAP. Isso não é algo ruim ou um demérito da linguagem, já que ela foi criada exatamente para isso. Entretanto, ter a consciência de que todo o portfólio SAP é orientado para empresas é algo crucial para entender os motivos que fazem o mercado ser como ele é.

Por um lado, empresas e grandes corporações estão dispostas a investir altas quantidades de dinheiro para otimizar seus processos. Obviamente estou sendo bem simplista nessa frase, já que empresas **sempre** vão fazer o possível para enxugar seus gastos, mas o mercado SAP gera bastante dinheiro e uma parte razoável chega até os profissionais do meio.

Outros pontos positivos envolvem o trabalho em muitas empresas gigantescas, gerando uma experiência valiosa para a carreira do profissional, além de oportunidade de viagens nacionais e internacionais para trabalhar em projetos dos clientes.

ABAPers costumam transitar por vários projetos em diversas empresas, gerando um dinamismo enorme nos esquemas de trabalho, práticas adotadas e padrões de desenvolvimento. É preciso uma capacidade de adaptação rápida para manter-se produtivo. O cliente X pode ter um *guideline* completamente engessado de trabalho, inibindo em muito a capacidade do ABAPer agir. Já o cliente Y pode não se importar em controlar absolutamente nada, deixando cada desenvolvedor implantar os programas do jeito que quiser.

Os relacionamentos no mundo SAP costumam ser extremamente importantes e são a base para contratos de trabalho e oportunidades. Dadas as dificuldades de entrada que já abordamos, é natural que o número de profissionais seja relativamente pequeno. Nesse cenário, fica fácil conhecer uma quantidade considerável de pessoas que podem lhe ajudar quando preciso, não só com ABAP, mas também outras áreas relacionadas ao SAP ERP (negócios e infraestrutura).

Só que essas empresas que gastam rios de dinheiro na customização de seus sistemas contêm uma urgência recorrente em seus requerimentos, gerando uma carga de trabalho muito grande e uma enorme oferta de empregos (que nem sempre são preenchidas). Essa urgência também faz com que a concorrência entre consultorias para ganhar um determinado contrato seja bem acirrada, em processos extremamente lentos e burocráticos, como veremos no capítulo 4.

Essa forma de fazer as coisas correndo também prejudica completamente a formação dos profissionais: se tudo é para ontem, não há tempo para aprender direto, somente tempo para entregar. Como o ABAP abstrai e simplifica diversos problemas (acesso ao BD, telas para seleção, relatórios), a maior parte do tempo costuma ser gasta tentando implantar regras de negócios muito complexas e fazendo sistema e dados trabalharem da forma que o cliente pediu. É preciso muito cuidado para esse complexidade funcional não fazer com que as *skills* de programação fiquem de lado, gerando "tradutores de regras de negócio", e não programadores.

Encontrar programas com mais de 10 anos em ambientes produtivos que são cruciais para o funcionamento da empresa é algo completamente normal para um ABAPer. Algumas áreas externas ao SAP acabam deparando-se com códigos mal-estruturados e complexos de serem assimilados, sem entender

completamente esse contexto do software corporativo criado em cima de um sistema com mais de 30 anos nas costas. Isso cria uma distância ainda maior por parte de alguns programadores, por acharem que **todo** e qualquer desenvolvimento ABAP é tratado dessa forma, o que não é real.

2.3 Problemas ou oportunidades?

Software corporativo dificilmente será tratado da mesma maneira que software para as grandes massas. São mundos completamente diferentes, cada um com as suas particularidades. Todos os pontos que citei podem ser interpretados como problemas pelas pessoas, já que é um cenário que pode não parecer muito atrativo.

Mas o que você acharia de criar um sistema analítico que calcula e otimiza as rotas logísticas de uma multinacional? Ou criar a automatização de processos entre sistemas distintos para movimentar mais de 5 mil transações por minuto? Ou mesmo criar relatórios extremamente complexos de tomada de decisão, que podem movimentar dezenas de milhões de registros (e dinheiros)? Criar monitores unificados para automatizar a criação dos documentos envolvidos, em praticamente qualquer negócio da empresa?

Todos esses cenários envolvem desenvolvimentos complexos que necessitam de um *expertise* técnico alto. Não é só conhecimento ABAP, envolve conhecer o *landscape* do cliente, como integrar os dados, como unir os sistemas, como padronizar comunicações ou como otimizar tecnicamente programas que processam quantidades gigantescas de dados.

O mundo SAP contém muitas oportunidades de projetos interessante e complexos para aqueles que possuem os conhecimentos adequados, um número de pessoas que você já deve ter deduzido não ser tão enorme, dado o cenário que expliquei até agora. São oportunidades reais de resolução de problemas complexos, que trazem uma grande satisfação quando concluídos.

Na minha visão, apesar de ser um mundo fechado e existirem diversas barreiras de entrada, uma vez dentro do mundo SAP, todo problema se transforma em uma grande oportunidade. Muitas consultorias e até mesmo clientes subestimam diariamente a capacidade que eles têm em mãos, com todas as ferramentas disponibilizadas pela SAP. Se usadas com maestria, é possível

transitar entre as mais diversas tecnologias (incluindo produtos/linguagens não-SAP) e criar aplicações muito robustas.

É uma situação muito parecida com a das empresas mais novas (aquelas startups com escritórios de frente para o mar), onde quase tudo é possível, basta alguém tomar a iniciativa e decidir implantar mudanças. Não que o caminho seja fácil, afinal, empresas grandes podem se tornar buracos negros burocráticos. Mas, se fosse fácil, também não teria muita graça.

Essas oportunidades ficam ainda mais evidentes se considerarmos toda a movimentação da SAP nos últimos anos para repensar o desenvolvimento de software na empresa, adotando práticas como UIs utilizando bibliotecas Javascript (SAPUI5), e serviços para padronizar e facilitar o consumo das informações (ODATA via Gateway ou HANA). Tanto que existe até um GitHub oficial da empresa, disponível em https://github.com/sap. Falaremos de tudo isso e mais um pouco no capítulo 8.

2.4 SAP é o mundo perfeito

Mas **nem de longe**. Para começar, pense na quantidade de funcionários da empresa envolvidos na utilização de um software, como o SAP ERP. Uma área de TI de uma multinacional pode ter times muito grandes, normalmente espalhados por várias partes do globo. Essa é a maior dificuldade de encontrar inovadores e entusiastas nesse meio: atingir um consenso entre profissionais e finanças na empresa, para conseguir trabalhar com práticas atuais, sistemas atualizados e equipes capacitadas.

Um ABAPer verá de tudo, desde aquela empresa que não tem dinheiro para investir em atualizações e trabalha com processos jurássicos até aquela outra que usa e abusa de todas as atualizações *patches*, correções e ferramentas que uma licença SAP disponibiliza.

Por mais tentador que seja trabalhar somente nas empresas inovadoras, é de extrema importância ter conhecimento de sistemas ou processos antigos, para que você possa entender o porquê eles **não** eram a melhor solução. Contanto que você não assuma que esses formatos e mecânicas falhas de trabalho sejam a regra, é possível fazer bom proveito das experiências.

O ABAP é relevante? Sim, e ainda vai se manter no mercado TI por muitos

anos. Afinal, a mesma burocracia existente para trabalhar com um sistema SAP ERP também está presente em uma decisão de troca, ou inutilização de um sistema tão gigantesco.

Se ele se tornará uma linguagem esquecida em um sistema "encostado", ou continuará sendo relevante para o ecossistema a longo prazo, é impossível prever, pois depende diretamente da aceitação dos clientes para as novas tecnologias lançadas pela SAP. Por se tratar de uma empresa desse porte e com essa penetração de mercado, é improvável que ele suma tão cedo.

Claro que nada dura para sempre, mas, mesmo que o ABAP vá para o vinagre, ainda existirão diversas oportunidades no mundo SAP, tanto para desenvolvedores como para o resto dos profissionais envolvidos. Basta manter-se atualizado, e não subestimar a sinergia e capacidade de integração do ABAP e ferramentas SAP com todo e qualquer outro tipo de tecnologia. Ser "somente" um ABAPer não é o suficiente.

Capítulo 3

Certificações ABAP

Se a sua intenção é a de começar como ABAPer e você não tem nenhum tipo de envolvimento inicial em empresas (estágio, trainee etc.), há grandes chances de você ter chegado a alguma informação relacionada às certificações SAP. Mesmo aqueles que já estão na área devem ter se perguntado, em algum momento, se a certificação pode ser um meio interessante de alavancar o currículo.

Os valores não são baratos, principalmente por você ser obrigado a fazer o curso oficial para poder prestar a prova (isso é uma barreira existente no Brasil, mas não é assim em outros países). Certamente, antes de investir rios de dinheiro, você deve ter se questionado: qual o real valor da certificação na prática do mercado? Essa resposta não é tão simples assim de ser respondida, apesar de os milhares de *banners* em sites de vendas de cursos tentarem provar o contrário.

Esqueça o mercado: quanto vale uma certificação para você?

Antes de discutir como o mercado enxerga as certificações, destaco que existem pessoas que enxergam a certificação como uma forma de objetivar seus estudos em determinada tecnologia. Nem todo mundo consegue estudar por conta própria e sem metas. Algumas pessoas precisam de uma motivação para focar em seu desenvolvimento.

Para essas pessoas, não importa tanto assim o valor que o mercado dá para a certificação, importa mais a experiência adquirida durante o processo de estudos para a prova final. Normalmente, elas não vão estudar somente o que é necessário para passar na prova – na verdade, ser aprovado é consequência do seu interesse a respeito do tema.

Se você tem essa forma de pensar, não pare para refletir sobre o valor que isso terá para o mercado. Vá em frente e coloque seu foco nas certificações. Tenho certeza de que o processo todo será benéfico para você.

Para todos os outros casos...

3.1 Certificação e as consultorias

Desde o início do meu trabalho com SAP até a publicação deste livro (9 anos+), nenhum líder, gerente, sócio ou cliente nunca mediu o meu conhecimento com base em uma certificação. Mudei algumas vezes de empresa, e as avaliações sempre se basearam na minha experiência e conhecimentos técnicos que demonstrei durante os processos seletivos.

Um leitor atento pode pensar: "Ué, mas como você vai saber se a certificação realmente pesou, se você não tem certeza absoluta dos motivos que fizeram você não ser contratado em algum processo seletivo?".

De fato, não tenho como saber os motivos que fizeram com que eu não passasse em algumas entrevistas. Entretanto, é possível ter uma ideia ao conversar com recrutadores, além de balizar pelas minhas próprias experiências como avaliador durante o recrutamento.

A certificação demonstra que o profissional dedicou parte do seu tempo aos estudos e conseguiu passar em uma prova. O problema é que a prova está longe de refletir as necessidades do dia a dia, portanto, não dá para utilizar o

título "*Certificado em XYZ*" como atestado de proficiência técnica. Já participei como avaliador de alguns processos seletivos, e a minha percepção inicial continua válida: para as consultorias, a experiência conta muito mais do que qualquer certificação. Uma avaliação técnica é sempre mandatória para conseguir medir ao menos parte do conhecimento dos candidatos, sendo ele certificado ou não.

Curiosamente, a maior parte das pessoas certificadas que conheço fez o teste de certificação quando estava iniciando sua carreira, utilizando-o como um mecanismo para provar que tinham um diferencial em comparação aos outros juniores. Só que isso levanta outra dúvida: a certificação não deveria ser um mecanismo para validar que determinado profissional atingiu um nível de conhecimento elevado, além dos temas mais básicos?

3.2 Os cavaleiros da certificação: Certification 5

Por conta de todas essas constatações, uma equipe de SAP Mentors (título entregue a membros de destaque da comunidade pela própria SAP, parecido com o programa MVP da Microsoft) uniu-se há alguns anos para criticar o formato das certificações no mundo SAP. Não pense você que são só os profissionais brasileiros que têm diversas dúvidas sobre esse processo.

Avaliando discussões no SCN, fica claro que o mundo inteiro partilha das mesmas dúvidas, questionamentos, indignações e incertezas!

Vale mesmo a pena ser um profissional certificado? A certificação conta mais do que a experiência na hora da contratação? A prova de certificação é aplicada da maneira correta? Os testes realmente conseguem avaliar o nível de conhecimento do profissional? Estes são só exemplos das milhares de dúvidas que pairam pelo mundo SAP quando falamos de certificações.

Muita discussão rolou nos últimos anos. Assim, cinco SAP Mentors uniram-se e criaram o *Certification 5*, grupo que fez uma crítica elaborada e extensa às certificações em um artigo público de mais 50 páginas (http://bit.ly/2c9Y5ZH) , publicado em 2010.

A ideia principal foi discutir a melhor forma de aplicar o exame de certificação, para que ele pudesse ganhar o devido valor no mercado. No artigo,

eles dão diversas sugestões sobre como a reestruturação do exame de certificação SAP pode trazer benefícios a todos da comunidade: clientes, profissionais e até a própria SAP. O grupo também fez uma pesquisa extensa que listou as percepções de diversos consultores do mundo todo a respeito de certificações. Dennis Howlett, um dos SAP Mentors que criou o artigo original, cita em um de seus artigos sobre o tema (http://bit.ly/2bC7Ydi) que um dos motivos que levou à criação foi a vontade de que uma certificação SAP tenha

o mesmo peso que um certificado de um médico, dentista ou advogado. Ou seja, que seja uma marca garantida de qualidade profissional.

Ele pondera que o peso da certificação é necessário, por conta do esquema da carreira dos consultores envolvidos com SAP. Normalmente, esse é um trabalho para a vida toda, não sendo um mercado com uma massa grande de profissionais que trabalham alguns anos na área, para depois mudarem para outras tecnologias.

É interessante analisar também que a equipe do *Certification 5* sabia, desde o início, que seria muito difícil para a SAP mudar todo o seu esquema de certificação para algo melhor, dada o tamanho da empresa e a complexidade de reestruturar todo o sistema.

No PDF do artigo, você pode ainda ver as respostas que a própria divisão de certificações da SAP deu para as críticas levantadas pelo grupo. É interessante ver o posicionamento da SAP Education para algumas questões. Isso ajuda a entender a mecânica do processo, enaltecendo a dificuldade em adequá-lo para o que entendemos como sendo satisfatório.

Analisando todo o material, fica evidente que não existe a verdade absoluta sobre o real peso de uma certificação do mercado SAP. Há muitas dúvidas, partindo de todas as áreas: clientes, consultores, consultorias e até de dentro da própria SAP (como visto nos comentários do artigo).

O *Certification 5* levantou questões que estavam "na garganta" de muitos profissionais do mundo SAP, porém, ao menos na área ABAP, pouco mudou desde a movimentação. Os questionamentos, dúvidas e confusões sobre o valor desse papel ainda persistem e são constantemente abordados na nossa área, principalmente entre aqueles que ainda não têm uma vaga dentro desse mercado.

3.3 Certificação e clientes SAP

Apesar da minha perspectiva nada otimista, acredito que nem tudo está "perdido" para as certificações no formato atual. Dentro desse cenário cheio de incertezas, existe uma situação onde a certificação revela-se necessária: exigência do cliente.

Alguns projetos em empresas gigantescas exigem que a consultoria apresente um número X de profissionais certificados, para que a consultoria possa entrar em um processo de concorrência.

Quando trabalhei em consultorias multinacionais, era comum chegarem e-mails da área do RH de tempos em tempos, tentando caçar profissionais certificados, provavelmente para submeter seus currículos para alguma dessas concorrências. Note que isso não queria dizer que o profissional certificado seria alocado no projeto, caso a consultoria ganhasse. O importante era cumprir as exigências do cliente, para poder entrar no processo de concorrência e submeter uma proposta de trabalho. A impressão que isso passa é da certificação ser uma mera formalidade e nada mais.

E aí, vou ou não vou?

Como disse anteriormente, tudo depende da sua motivação. Não faça a certificação simplesmente *porque sim*, ou porque alguém está sugerindo que isso será um diferencial matador. Você corre o risco de gastar uma grande quantidade de dinheiro, e não obter o retorno esperado.

Tenho certeza absoluta de que profissional nenhum gosta de gastar dinheiro à toa. Afinal, os valores não são baixos. Portanto, avalie bem o que a certificação significa para você, antes de entrar nessa jogada. Vá, mas vá com cautela!

Capítulo 4

Os bastidores

Neste ponto, já ficou claro que os grandes empregadores do mundo SAP são as consultorias. São elas que negociam os projetos com clientes, e decidem quando e onde o desenvolvedor vai pôr em prática aquilo que sabe.

Essa dinâmica gera as mais diversas polêmicas e discussões, já que muitas vezes as práticas das consultorias podem parecer um tanto quanto abusivas, levando o profissional a sentir-se explorado e descontente com o seu trabalho.

O que eu notei ao longo dos anos é que muitos dos ABAPers que reclamam de as consultorias serem exploradoras sequer entendem o básico de como uma consultoria funciona. Pode parecer simples, mas há uma série de profissionais que precisam interagir até que o ABAPer seja alocado no projeto X ou Y.

Neste capítulo, explicarei a partir das minhas experiências um pouco da

mecânica de como as coisas funcionam, para lhe ajudar a decidir se a consultoria para a qual você trabalha está **mesmo** sendo abusiva. É claro que há gerentes odiosos, líderes carrascos e comerciais tapados. Mas, às vezes, tudo o que precisamos é entender o contexto que nos levou àquela alocação bizarra. Afinal, esse povo bizarro e que trabalha mal pode ser identificado facilmente, não é mesmo?

Importante: as explicações deste capítulo foram criadas do ponto de vista das consultorias, e não envolvem a mecânica de trabalho dos ABAPers contratados como funcionários de uma empresa que contém produtos SAP. Normalmente, esses desenvolvedores são escalados para cargos relacionados ao suporte diário e aos problemas de produção, que envolvem a correção de erros e melhorias específicas em alguns processos.

Por conta do fato de as dinâmicas desses trabalhos serem completamente distintas (depende muito da cultura e prática de cada empresa), elas não serão consideradas neste momento. Mas, se você for um desses ABAPers, não pule o capítulo: há grandes chances de você considerar um emprego em uma consultoria, em um período de "dúvidas" da sua carreira. Saber como as coisas funcionam facilitará a sua possível transição.

4.1 Como surgem as alocações?

Todo mundo gosta de falar que o SAP ERP é um produto extremamente caro, mas o valor pago pela licença costuma ser muito inferior à quantidade de dinheiro gasta com consultorias. É aquela velha história de que implantar o SAP ERP por si só não é o bastante. É preciso também customizar o sistema: adequando regras de negócio, criando relatórios complementares, aumentando a produtividade da equipe, ou até mesmo trocando a cor de uma tela porque algum *bam-bam-bam* quis assim. E tudo isso, obviamente, custa (muito) dinheiro.

A necessidade de um ABAPer nasce muito antes do recebimento de uma especificação funcional, ela tem início na mente do cliente. A partir do momento em que um cliente decide que ele precisa ajustar/melhorar/criar uma determinada funcionalidade, o papel do ABAPer pode tornar-se necessário. Todo trâmite envolvido nesse processo da ideia à realização é que determi-

nará como e quando será a alocação do desenvolvedor e outros profissionais do mundo SAP.

Funciona assim: quando o cliente enfim decide que ele **realmente** precisa alterar o sistema de alguma forma (isso pode demorar anos, acredite), ele vai contatar diversas consultorias para que elas façam um orçamento de quanto ficará o trabalho. Elas precisarão avaliar o requerimento do cliente e definir uma solução técnica. Isso gerará uma estimativa de esforço para poder quantificar o valor das alterações/criações, estas sumarizadas em um documento que as consultorias chamam de **propostas**. São nelas que estão listados o escopo do trabalho, o número de profissionais envolvidos, a estimativa de hora para os desenvolvimentos e o valor cobrado pelos serviços prestados.

Muitas palavras "corporativas" no último parágrafo? Vou reescrever para ficar mais claro para a nossa realidade: o cliente quer adicionar alguma doideira ao sistema. Então, a consultoria verifica se ele não está viajando e define uma solução técnica, tentando dar um chute certeiro do tempo que será gasto nos trabalhos, contando com folgas nos tempos e prevendo alterações do escopo (o cliente **sempre** pede algo mais, **sempre**). Isso tudo é sumarizado em um arquivo Word/Excel – que muitas vezes é grande demais para ser entendido por um humano –, e é entregue ao cliente para que ele avalie se a solução, a estimativa de tempo e os valores da consultoria fazem ou não sentido.

Munido das propostas de diversas consultorias, o cliente vai decidir com qual seguirá em frente. Apesar de essa decisão parecer algo determinístico (deve ganhar a melhor proposta, certo?), essa fase pode ser completamente subjetiva.

A situação é bem parecida com a de quando você decide que comprar na *padaria do Zé da esquina* é melhor do que ir à *padaria de superqualidade* que fica a 200 metros de sua casa, só para evitar a fadiga. Essa decisão costuma ficar ainda mais fácil se o Zé for seu parente: certamente você só vai comprar com ele... até que você compre algo estragado, o que fará com que você possa voltar a considerar a *padaria de superqualidade* (ou não, vai saber).

É importante notar que algumas consultorias podem fazer os mais diversos malabarismos para ganhar um projeto de um cliente onde há uma disputa acirrada. Dependendo do malabarismo, os profissionais alocados no projeto podem sofrer muito. Desde despesas não pagas até milhares de horas extras

para cobrir um erro na venda.

Aliás, esses problemas na venda geram a reclamação número um dos projetos atrasados ou corridos. Isso é natural, pois se o projeto nascer errado, as chances de todos os envolvidos sofrerem muito com sua implantação são altíssimas.

Pois bem, esse trâmite todo que envolve gerentes, líderes, sócios, comerciais e especialistas pode demorar um tempão para terminar. Tudo depende do escopo do trabalho: quanto maior o projeto, mais demorado ficará o processo da criação das propostas.

Aquelas reuniões intermináveis com os especialistas das consultorias são muito comuns nesta fase, já que o cliente pode não ter noção da quantidade de horas que podem ser gastas para ajustar algo que ele pensa que é simples. Muita saliva costuma ser gasta até alguém conseguir explicar, de forma consistente, que algo que deveria custar 10, na verdade custa 1.000. Há ainda diversas rodadas de perguntas e repostas, tudo para que a consultoria possa chegar à melhor relação de solução e preço.

Apesar da demora e da complexidade do trabalho, hora ou outra o projeto pode ser fechado com a consultoria em que você trabalha (sejam lá quais forem as condições acordadas). E é nessa hora que é preciso decidir algo extremamente crucial: quem da equipe vai trabalhar no projeto?

4.2 A dança das cadeiras

"Ah, essa é outra decisão fácil, não é mesmo? Basta escolher quem estiver disponível e mandar para o projeto!

Quem dera essa decisão fosse assim tão simples.

O dinheiro da consultoria vem da alocação dos seus profissionais, portanto, consultoria nenhuma tem interesse em deixar pessoas desalocadas. A venda constante dos seus serviços é essencial para manter seus profissionais faturando em clientes. Sem isso, consultoria nenhuma sobrevive.

O maior problema na hora de decidir quem vai para o projeto é que nem sempre as datas coincidem. Não dá para chegar para o cliente e falar: "sabe aquele projeto que vai resolver todos os seus problemas e que você quer fechar conosco? Então, vamos deixar para o ano que vem, pois não tenho ninguém

disponível para atendê-lo".

Dizer algo assim é basicamente dar um atestado para o cliente dizendo "feche com outra empresa". Em situações específicas (projetos ruins ou para evitar a superalocação dos consultores), recusar um projeto pode ser a saída. Porém, toda consultoria tem interesse em fechar projetos interessantes de grandes empresas, tendo ou não os profissionais para realizar os projetos.

Ué, mas peraí! Como assim as consultorias fecham projetos sem ter pessoas para trabalhar neles?

É isso mesmo: como as datas entre os projetos que estão rolando na consultoria podem não coincidir com as datas dos novos, a consultoria pode fechar projetos mesmo sem saber quem vai trabalhar neles. No melhor dos mundos, ela consegue remanejar seus profissionais para atender às novas demandas, mas muitas vezes ela precisa recorrer ao mercado e contratar novos funcionários.

Essa situação, aliás, é a que mais gera aquela avalanche diária de ofertas de vagas que vemos todos os dias em grupos de e-mail, Facebook, LinkedIn e afins. A consultoria pode, inclusive, começar a abrir as vagas sem nem mesmo saber se o acordo com o cliente será realmente fechado, como um mecanismo de precaução caso ganhe o projeto. Isso não é uma prática exclusiva do mundo SAP; qualquer área de TI relacionada a serviços acaba caindo em situações similares.

Supondo que a consultoria tenha as pessoas necessárias para atender a demanda, fica a mesma dúvida anterior: quem da equipe vai trabalhar no projeto? Em todas as consultorias em que trabalhei, os melhores profissionais tinham preferência para os melhores projetos.

Se for um cliente estratégico – ou seja, que tem muito dinheiro para gastar –, pode até ser que um ótimo ABAPer seja substituído no projeto onde está, para que possa trabalhar no novo projeto importante. É exatamente por isso que é extremamente essencial manter-se atualizado, estudar de forma contínua e procurar realizar sempre um ótimo trabalho. Bons gestores precisam de pessoas competentes em quem possam confiar, e essa pessoas terão prioridade em escolhas como esta.

Particularmente, demorei um pouco para entender essa dinâmica, pois um ABAPer no início da carreira tem pouquíssimo contato com esse mundo

das propostas e contratos fechados de projetos. Mas depois de entendê-la, percebi que muitas das pessoas que pegam os piores projetos e alocações acabam sendo pessoas que não são escolhidas com prioridade pela consultoria.

Uma analogia boa é o futebol: em um jogo de decisão, quem joga no time titular? E quem entra em um jogo que tem pouquíssima importância, mas precisa ser cumprido? Se o jogador *mega master blaster* está lesionado, quem vai para o seu lugar? Será que é mais vantajoso contratar um novo jogador em vez de pegar alguém do banco de reservas?

Pare e pense: em uma situação similar, se você fosse o gestor, quem você escolheria?

4.3 Mecânica de trabalho

Após quilos e litros de arquivos do Word e Excel, inúmeros e-mails com 384 pessoas em cópia e uma infinidade de "reuniões de alinhamento", sua consultoria enfim ganhou o projeto e decidiu que você será o alocado da vez. E agora?

No início dos projetos, você provavelmente vai receber uma série de documentos que dizem quais padrões você deve seguir ao programar para aquele cliente. São padrões de nomenclatura, diretrizes de performance, templates de especificações técnicas e roteiros para testes unitários. É claro que isso só acontecerá em projetos organizados. Na maioria deles, é você que precisará perguntar por estes documentos. Dica de amigo: faça isso logo que chegar, para não correr o risco de fazer as coisas "do seu jeito", e ter que alterar tudo depois de pronto (como já aconteceu comigo, mais de uma vez).

Agora que você já está a par de todas as normas e diretrizes, resta uma corrida constante contra o tempo. Durante a proposta, algumas estimativas foram estipuladas, e elas diziam que o programa ABAP X iria demorar, mais ou menos, o tempo Y para ser realizado.

"Mas isso é um absurdo!", diz o ABAPer revoltado. Na maioria das vezes, as pessoas que se envolvem com propostas são especialistas ABAPers, portanto, os tempos não devem estar totalmente fora da realidade. Se, mesmo assim, alguém chegar com prazos completamente apertados, lembre-se daqueles malabarismos de que falei há alguns parágrafos atrás. Nem sempre

os especialistas ABAPers estão presentes na hora de definir qual desenvolvimento será penalizado "em prol da venda do projeto".

Consultorias grandes dispõem de uma fase em que a estimativa feita, em tempo de proposta, deve ser confirmada, já que é na hora do projeto que a empresa contratante vai liberar todos os detalhes necessários de uma determinada demanda. Nem sempre isso existe em consultorias menores; todos os prazos já foram predeterminados e pronto. O ABAPer pode até tentar discutir com o gerente para ver se uma demanda que precisaria do dobro de tempo para ser concluída pode ser remanejada, mas isso nem sempre é possível.

O cliente pode, inclusive, inventar novas funcionalidades "do nada" e, dependendo do caso, a consultoria precisa arcar com o prejuízo. A realidade é bem mais cruel do que aquilo que é ensinado pelos professores de engenharia de software.

Durante sua carreira como desenvolvedor, você verá diversos artigos, podcasts e livros (este inclusive) dizendo para não ir pelo caminho mais fácil na hora de programar. Em uma situação de prazos apertados, como fazer o melhor desenvolvimento? Acredite quando eu lhe digo que é **sempre** melhor fazer as coisas direito, por mais curto que seja o tempo.

Você pode até gastar um tempo maior desenvolvendo, mas o número de problemas que você vai evitar no futuro valerá a pena. Basta você convencer o seu gerente de que isso faz sentido. Se ele não acreditar, uso o bom e velho "eu avisei" quando determinado programa ruim, gerado pelo projeto, atrapalhar completamente o teste integrado. Um "eu avisei" bem empregado sempre funciona. Só tome cuidado com as reações!

Mas acalme-se, nem tudo é tão caótico. Muitos projetos contêm prazos muito maiores do que o necessário. Isso é fruto da famosa "gordura", um tempo adicional que a consultoria adiciona às propostas, prevendo atrasos ou alterações de escopo pela parte do cliente. Se você já pensou em "por que determinado ABAPer fica o dia inteiro sem fazer nada", esse tipo de demanda é uma provável resposta.

O prazo estipulado para as demandas não deve ser utilizado somente para a criação do código, você também precisa testar corretamente a sua aplicação e entregar quaisquer que sejam os documentos que o projeto pedir. A especificação técnica costuma ser o documento mais ignorado pelos ABAPers – e

falaremos mais dela nos próximos capítulos.

Os testes dependem muito dos cenários gerados pelo time funcional e pelo cliente, mas eles precisam ser feitos. Não hesite em encher o saco de ambos a fim de que eles criem os cenários necessários para que você possa validar, se a sua aplicação realmente faz aquilo que deveria fazer.

No fim, quase tudo gira em torno do tempo nos projetos. Prepare-se para lidar com a pergunta que eu mais odeio nesta carreira: "e aí, quanto tempo falta para você terminar essa demanda?". Aposto que o SAP ERP fará o possível e imaginável para que a sua resposta ("só mais um dia") seja completamente falha. Se envolver `BAPI`s, alterações em programas antigos, ampliações de processos nativos do sistema ou `CALL TRANSANCTION`s, a chance de sua resposta ser furada é ainda maior.

Assim como acontece na venda, por maior que sejam os problemas que um projeto possa gerar, hora ou outra ele entrará em produção, por meio da fase conhecida como *Go Live*. Eu gosto de dizer que é nesse momento que aqueles que não trabalharam direito pagam os seus pecados.

Os desenvolvimentos vão todos para a produção, e aqueles que não foram feitos (ou testados) corretamente correm grandes riscos de gerarem perdas monetárias reais para a empresa contratante. Não são raras as lendas urbanas de programas ABAP que atrapalhavam determinado processo e geraram perdas imensas para linhas de produção dos clientes.

A famosa frase "os caminhões estão parados" surgiu de programas relacionados à liberação do caminhão (Nota Fiscal etc.), que simplesmente não funcionavam em produção. Tenho amigos que já ficaram com a orelha vermelha por conta de caminhoneiros que esperavam um dia inteiro na porta da empresa, até que alguém arrumasse "o sistema".

É, amigo ABAPer, não subestime a nossa capacidade de atrapalhar a vida dos outros através de códigos falhos. Preocupe-se em entregar um trabalho correto e você não sofrerá dores de cabeça. Assim, mesmo que algo que você fez dê errado, seus pares saberão que não foi por falta de empenho e farão o possível para ajudá-lo a corrigir as falhas.

4.4 E então vem a próxima alocação

O *Go Live* foi um sucesso, todos os programas estão em produção, o sistema aparentemente não tem mais falhas e a sua alocação está perto do fim. É nesse momento que o ciclo se repete.

De volta à consultoria, gestores vão avaliar quais os projetos disponíveis, para poder remanejar sua equipe. Há algum projeto novo precisando de pessoas? Há algum projeto corrente que precisa de ajuda? Sendo você parte da equipe, seu futuro dependerá diretamente das decisões do gestores sobre onde seria o melhor lugar para o alocar.

Isso pode acontecer diretamente, por exemplo: você sai na sexta-feira de um projeto para começar na segunda-feira em outro completamente diferente, ou pode demorar algum tempo para acontecer, em um momento em que você se encontrará "desalocado".

A desalocação é vista de forma ruim pela maioria das pessoas do mundo SAP. Mas como já expliquei, nem sempre as datas dos projetos coincidem, o que leva a consultoria a precisar deixar um profissional parado por um tempo, até que um novo projeto inicie. Procure saber claramente se a sua desalocação é decorrente de uma problema de datas, ou por que o seu gestor preferiu mandar uma outra pessoas em seu lugar. É sempre bom saber o motivo real das coisas, por pior que ele seja.

Independente do motivo, se você encontra-se desalocado, terá de achar uma resposta para a derradeira pergunta: o que fazer? Redes sociais, jogos e assuntos pessoais vão fazer de tudo para tomar o seu tempo. Resista!

Converse com seu gestor e veja se ele precisa de ajuda. Se ele não precisar, aproveite esse tempo livre para estudar ou revisar "aquela" coisa que gerou problemas no último projeto. Use o tempo com algo útil e demonstre interesse. Ficar parado esperando alguém decidir o seu futuro é provavelmente a pior forma de aproveitar esse tempo.

"Ah, mas um descanso faz bem...". Certa vez, reclamei de cansaço e férias para a faxineira do escritório onde trabalho, dizendo que eu estava há mais de 2 anos sem férias. Ela respondeu para mim com enorme simplicidade: "Eu sei como é, meu filho, estou há 8 anos sem férias". Ela acorda todo dia 04:30, chega em casa às 22:00 e é uma das pessoas mais alegres que já conheci.

É claro que eu acredito que tirar férias anuais é algo muito bom (e eu recomendo fortemente que você faça isso), mas o ser humano realmente gosta de reclamar de barriga cheia. Tem um tempo livre no projeto? Tente utilizar para algo útil e demonstre isso para seus pares. Aposto que você colherá bons frutos.

Voltando ao período de transição entre projetos, é muito importante que você constantemente entenda a sua posição dentro da consultoria. Uma coisa que é frequentemente ignorada é o momento do feedback.

Durante um projeto, você certamente trabalhou com muitas pessoas dos mais diferentes cargos e funções, tudo para que você conseguisse entregar seus programas ABAP com qualidade. Será que eles gostaram do seu trabalho? Será que eles teriam alguma observação para fazer? Pedir feedback para os outros é uma forma ótima de ter certeza se você está ou não no lugar onde gostaria de estar.

Os feedbacks mais importantes são os dos seus gestores e dos seus amigos ABAPers mais experientes que já trabalharam com você. Eles podem revelar problemas que estejam fazendo com que outros avaliem mal o seu trabalho, ou pontos fortes que você pode usar para alavancar sua carreira.

Como a dinâmica das alocações pode fazer com que você saia de um projeto e vá diretamente para o outro, sem tempo para respirar, é importante pedir um feedback *antes* do fim do projeto. Não deixe para depois.

Quando você é alocado no próximo projeto, o ciclo se repete. Ele vai se repetir muitas e muitas vezes, até que você vire um gestor, um líder dentro dos projetos, um especialista (que fatalmente precisará ajudar em propostas), ou vire alguma outra coisa que eu não posso imaginar, decida que essa vida maluca não é para você ou continue sendo jogado de projeto em projeto como um tapa-buraco de alocações ruins de consultorias. A escolha é sempre sua.

Capítulo 5

Carreira, motivação, universo e tudo mais

Mudanças na carreira ABAPer podem demorar para acontecer. Você codifica, codifica e codifica, muitas vezes sem uma perspectiva clara de onde chegará, além da tríade júnior-pleno-sênior. Para piorar, processos burocráticos do dia a dia das empresas, que costumam ser ainda mais complexos do que os explorados no capítulo 4, contribuem muito para desmotivar os desenvolvedores.

Nesse cenário pseudoapocalíptico, minha maior recomendação é: *não entre em pânico*. As carreiras meteóricas do mundo SAP costumam atropelar a noção de muitos conceitos, envolvimento com outros desenvolvedores e até mesmo uma noção de que tudo sempre está ruim. É preciso ter cautela.

Os textos a seguir são reflexões sobre temas que impactam a sua carreira

e a sua motivação, para trilhar o caminho do ABAPer. De conceitos básicos à venda de cocos na praia, vamos refletir se é a indústria que realmente não dá brechas para o desenvolvimento dos ABAPers, ou se é tudo uma questão de pontos de vista.

5.1 Será que todo ABAPer é mesmo um consultor?

Quando ingressei no mundo SAP, sempre contava aos meus familiares que eu trabalhava como programador ABAP. Porém, com o passar do tempo, percebi que praticamente ninguém do mercado (clientes, recrutadores, usuários etc.) utiliza o termo programador: o profissional ABAP é chamado de **consultor ABAP**.

Alguma vez você já parou para pensar no que há por trás dessa diferença?

Primeiramente, vamos analisar as definições de ambos os termos, diretamente do Dicionário Aurélio da Língua Portuguesa (versão online):

- **Consultor:**

  Consultor é o profissional que, por seu saber, sua experiência, é procurado para dar ou fornecer consultas técnicas ou pareceres, a respeito de assuntos ou matéria dentro de sua especialidade.

- **Programador:**

  Programador é o profissional especialista encarregado da preparação de um programa imposto a um computador.

Ou seja, **consultor** é aquele cara que consegue analisar a solução inteira para dar a melhor alternativa possível, para resolver um determinado problema; **programador** é parte daquela parcela mínima da população mundial que sabe escrever programas com instruções capazes de serem executadas por computadores, devidamente documentados e testados.

Se você pensar um pouco, é fácil chegar à conclusão de que todo programador experiente é também um ótimo consultor, mesmo não usando tal título.

A definição de que ABAPers são consultores e que profissionais Java/Ruby/PHP são programadores é uma mera prática de mercado, definida para justificar as altas taxas cobradas em meados dos anos 90, quando o SAP ERP começou a se popularizar. Por ser um sistema exclusivo para o mundo corporativo, um *mero* programador não era suficiente para implantá-lo, era preciso um **consultor**.

No mundo SAP, não existe um só profissional que entende as necessidades dos usuários, desenha os processos, programa, testa e implanta em produção. Esse papel é compartilhado entre os funcionais, que focam em implantar regras de negócio específicas da empresa nas regras já existentes no ERP; e os ABAPers, que supostamente deveriam ter bons conhecimentos sobre as necessidades dos usuários e do processo, para poder transformá-los em programas.

A figura a seguir demonstra a estrutura de um projeto SAP clássico, difundida e praticada por todo o mercado:

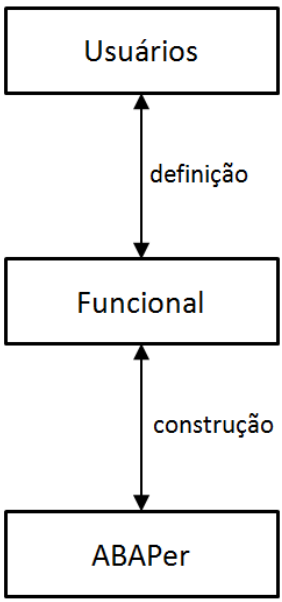


Fig. 5.1: Estrutura clássica de um projeto SAP

Nessa estrutura de projetos, quantas vezes o consultor ABAPer ajudou a definir a solução do projeto como um todo (e não somente a dos seus programas)? Quantas vezes foi envolvido em um *business blueprint* (fase da metodologia ASAP – Accelerated SAP – em que é criado o detalhamento dos processos, para que eles virem especificações funcionais)? Conseguiria ele responder, em poucas palavras, qual o propósito do seu projeto e quais impactos ele terá no dia a dia da empresa?

Em mais de 9 anos trabalhando com ABAPers, poucas vezes eu posso dizer que os vi realmente envolvidos na definição das soluções dos projetos. Essa situação é um pouco irônica: os consultores dizem que trabalhar com ABAP é mais do que fazer só a parte técnica, porém aposto que foram poucas as vezes em que eles realmente tiveram voz ativa nas definições de como um projeto foi implantado.

Todos acabam criando seus programas de acordo com o que foi pedido pelo funcional, sem saber exatamente do que se trata e qual a importância daquele desenvolvimento no projeto como um todo.

A impressão que tenho é a de que a definição de consultor não reflete em nada a realidade. No formato atual, os ABAPers ficam presos, enjaulados em um círculo vicioso de trabalho que destrói sua criatividade e a sua real capacidade de prestar consultoria.

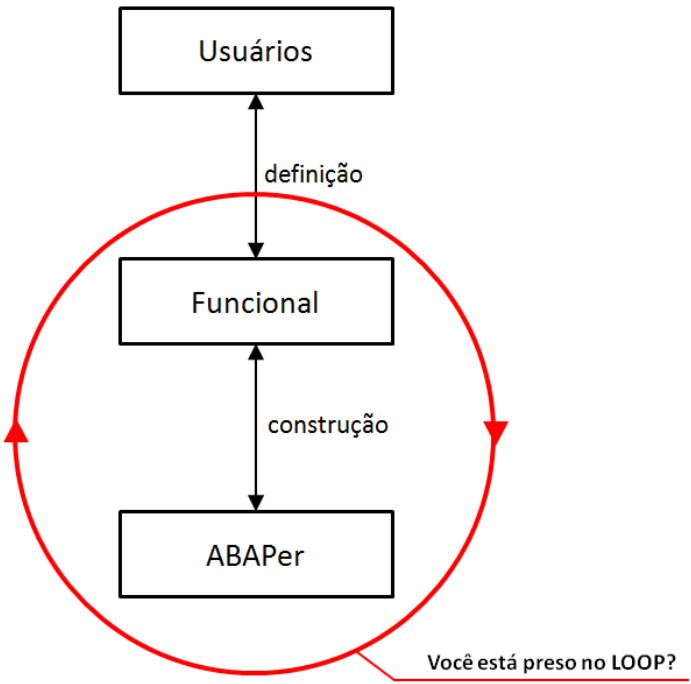


Fig. 5.2: LOOP eterno de um ABAPer sem lembraças

Pense: se os processos nos projetos fossem diferentes, e todos (funcionais, abapers, key users) pudessem desenhar juntos as soluções, o projeto seria implantado de forma diferente?

Uma das principais armas de um programador é a sua capacidade de abstrair os problemas e montar a arquitetura do sistema na sua mente, prevendo, assim, futuros problemas na parte técnica. Essa informação é extremamente valiosa em momentos iniciais de um projeto, e pode definir os rumos que ele vai tomar (para o bem ou para o mal).

Esse fluxo de trabalho não é ditado pelos ABAPers, é algo presente no mundo SAP há muito tempo. É um paradigma enraizado nas empresas.

O ABAPer que trabalha por intermédio de uma consultoria, alocado em clientes, normalmente precisa entregar seus desenvolvimentos no menor

tempo possível, sendo que o tempo para entender o contexto do projeto sequer é considerado. Essa falta de envolvimento com a parte de definição e arquitetura é fruto dos mais diversos problemas, desde sistemas com usabilidade terríveis até falhas estruturais que podem gerar atrasos ou, em casos graves, projetos perdidos (quando o cliente desiste de terminar um projeto).

Existe alguma forma de melhorar esse cenário?

Nosso mundo gira em torno das estimativas: quando o cliente precisa fazer alguma alteração/extensão do seu sistema, ele pedirá para a(s) consultoria(s) fazer(em) uma estimativa de tempo do trabalho, para avaliar qual será o custo da tarefa.

Existem diversas técnicas para descobrir o "número mágico" de horas que serão gastas com um desenvolvimento: abordagens científicas como a análise de pontos de função, que usam técnicas para estimar o tempo a partir das funcionalidades percebidas pelo o usuário; abordagens práticas criadas pela experiência de programadores, como as compartilhadas por Steve McConnell no livro *Software Estimation: Desmystifying the Black Art*; guidelines de estimativas (muito comuns em consultorias), que dirão que a funcionalidade ABAP X vai levar Y tempo para ser implantada; ou até mesmo o bom e velho "2 semanas", sem embasamento algum.

Independente da abordagem, esse é o momento perfeito para o ABAPer exercer, enfim, o papel de consultor: para estimar, é preciso imaginar **como** o sistema será implantado. Para arquitetar uma solução técnica, é necessário entender os requerimentos do cliente/funcionais, analisar as interfaces com outros programas, verificar os impactos em outros processos atuais, antecipar barreiras técnicas e limitações do sistema, avaliar se é possível simplificar o processo...

Às vezes, o parecer do ABAPer pode invalidar completamente a criação de um programa (por ser complexo demais), ou mesclar desenvolvimentos em prol da simplicidade (qual a necessidade de três interfaces novas, se já existem três interfaces no sistema que fazem praticamente a mesma coisa?).

Na minha visão, a melhor maneira de demonstrar a importância do trabalho ABAPer, nos momentos de definição, é não menosprezar as estimativas. Entender muito da linguagem para otimizar `SELECT`s é muito importante,

mas tão importante quanto é otimizar a arquitetura do programa durante as definições, removendo toda e qualquer seleção desnecessária.

É tentador "chutar" valores extremamente altos para requerimentos que serão trabalhosos, mas isso só contribui para que o ABAPer continue preso em seu `LOOP` com o time funcional, sem voz ativa e sem participação nos rumos dos projetos.

Em nossa prática diária, um título de consultor ou programador terá pouca importância. Mais importante é tentar envolver-se cada vez mais com as definições dos projetos por meio das estimativas, ou reuniões de *blueprint*. Isso fará com que o nosso conhecimento técnico possa ter impactos benéficos nos projetos, agregando valores grandes à solução entregue ao cliente.

5.2 Não seja atingido pelo despreparo alheio

Uma conversa recorrente em escritórios de consultorias e empresas que trabalham com SAP é sobre o despreparo de parte dos profissionais que oferecem seus serviços no mercado brasileiro. Mesmo com pouco tempo nessa área, você certamente escutará diversas histórias sobre profissionais *picaretas* e pessoas que não têm condições de trabalhar em projetos e/ou suporte.

É muito comum profissionais que trabalham com SAP conhecerem alguma pessoa que é popularmente intitulada de "mau profissional que sabe se vender". Caso você (ainda) não conheça, pense em um profissional que afirma ter conhecimentos sólidos sobre alguma tecnologia, mas acaba demorando horas infinitas para concluir tarefas básicas, além de não mostrar os tais conhecimentos sobre praticamente nada do que envolve seu trabalho.

Note que não estou considerando o despreparo de profissionais juniores recém-formados: o problema são os casos de plenos e seniores despreparados que coexistem com o mundo de consultorias, projetos e clientes há muito tempo.

Essas pessoas beneficiam-se das altas demandas e a baixa no número de profissionais. As consultorias sempre têm muita pressa na contratação (por pressão do cliente para começar os projetos), e vários desses *vendedores* entram de forma despercebida nos projetos. Até alguém descobrir que ele não sabe nada, o caos já foi instaurado, trazendo muita dor de cabeça e noites mal

dormidas.

Esse tipo situação está longe de acabar e, provavelmente, nunca será extinta. Filtrar na entrevista alguém despreparado para alguma coisa requer um nível de conhecimento que muitos recrutadores não possuem – e mesmo aqueles que o têm podem ter dificuldades no processo.

Contratar a pessoa certa não é uma tarefa fácil: todas as técnicas de recrutamento e seleção – como testes, avaliação comportamental, avaliação prática e dinâmicas – são meios de tentar conseguir o profissional correto – e nenhuma delas é infalível.

Piora o fato de que a pressa na contratação faz com que o processo seletivo pule algumas etapas e, muitas vezes, a prova técnica ABAPer é feita somente pela conversa (ou só por telefone). Se você já trabalhou para mais de uma consultoria deve ter se acostumado com o fato de não ser necessário fazer nenhum tipo de prova prática no processo seletivo para uma vaga de ABAPer.

O recrutador não avalia nenhum dos seus códigos, não vê você interpretando um requerimento do cliente, não vê você criando alguma solução para um problema... O que conta são experiências, respostas para perguntas técnicas "padrão" e uma eventual indicação. Três itens que deixam as portas abertas para qualquer um que saiba se vender.

Esse cenário joga contra a prática ABAPer, e o nivelamento do conhecimento necessário para alguém ganhar o título de bom profissional no mundo SAP, no geral, tende a ser muito baixo. Exemplo: se você trabalha em um projeto pequeno com 2 ABAPers, sendo que 1 deles é um ABAPer despreparado, você tem grandes chances de destacar-se sem fazer muita coisa, ganhando diversos elogios por parte do cliente, e de pessoas que não conseguem avaliar os detalhes técnicos por parte da consultoria (se eles não conseguem avaliar corretamente na contratação, por que avaliarão corretamente nos projetos?).

Aos poucos você cria a ilusão de que é um profissional *fora da curva*, e deixa de ter vontade de estudar e buscar conhecimento, por pensar que já sabe o suficiente para manter-se relevante para o mercado por muito tempo.

Essa situação pode acontecer de uma forma não explícita ao longo de sua carreira, e pode prejudicar muito o seu desenvolvimento profissional. Ela fará com que você entre ainda mais no seu "mundinho", onde tudo que você faz é perfeito e não é necessário aprender nada novo e nem se atualizar. Se você

consegue resolver os problemas daquele projeto e ainda é elogiado, para que mudar, não é mesmo? É assim que a onda do despreparo pode atingir-lhe, sem que você perceba.

5.3 Desvalorização do mercado

"Os ABAPers estão sendo desvalorizados! As taxas estão abaixando! Somos vítimas de consultorias mercenárias!". Escuto reclamações desse tipo desde 2006, quando comecei a trabalhar com ABAP.

Veteranos dizem que elas já existiam no final dos anos 90, quando o SAP ERP ainda estava se popularizando. E quem é que mais adora reclamar sobre injustiças salariais? Os mesmos despreparados que sobrevivem no ecossistema.

Por trás dessas reclamações, há um raciocínio bem simples: "Por que aprender algo novo, se existem tantos profissionais ruins e, com o pouco que eu sei, consigo ser bem remunerado? Por qual motivo preciso gastar tempo aprendendo técnicas e tecnologias atuais, se com o meu `START-OF-SELECTION`, `FORM BUSCA_DADOS`, `FORM FORMATA_DADOS` e `FORM MOSTRA_RELATÓRIO` (separação extremamente básica – e quase sempre falha – da lógica de um programa ABAP) consigo ganhar um bom salário?".

É esse tipo de gente que odeia compartilhar suas preciosas "manhas" (como se não existisse a internet), odeia ensinar iniciantes e fica irritada quando vê juniores discutindo conceitos que não são familiares entre os mais velhos. É como se a reclamação fosse um mecanismo de autodefesa, para não vazar o medo constante de serem passados para trás.

O que muita gente não percebe é que o nosso conhecimento como programador é o que trás lucro para todos os envolvidos no processo: clientes, consultorias e o próprio programador. Potencializando o conhecimento, é possível reverter o cenário e fazer com que o mercado tenha de pagar o seu valor em vez de você ter de reclamar de uma suposta desvalorização generalizada.

Certa vez recebi uma reclamação de que os artigos do ABAPZombie contribuíam para abaixar a taxa do mercado. Fico pensando, então, se esse livro

fará você perder seu emprego (atual ou futuro)...

Mantendo a motivação

É muito importante que você fique atento e não se deixe levar por reclamações irreais e um suposto senso de superioridade técnica, baseada na comparação com um possível trabalho medíocre dos seus pares.

Particularmente, gosto de usar o ensino como mecanismo para alavancar meu progresso e fugir desse cenário. Compartilhar conhecimento é mais do que uma forma de fazer "um nome" – essa é a parte menos importante. A responsabilidade de ensinar algo corretamente gera uma necessidade constante de pesquisar a fundo sobre os diversos conceitos, para compor artigos que entreguem informações de qualidade.

Isso funciona para mim e para muitos outros, mas pode não funcionar para você. É preciso que você encontre o seu próprio mecanismo. Gosta de buscar soluções para dúvidas? Gaste alguns minutos diários lendo o *feed* da comunidade ABAP do SCN, que agrega perguntas técnicas de pessoas do mundo todo. Prefere algo mais prático? Envolva-se com a criação de alguma ferramenta ABAP open source através do GitHub. Você não é do tipo que gosta de desenvolver em comunidade? Então melhore suas habilidades de programação, utilizando sites com problemas matemáticos feitos para serem resolvidos por programadores, como o *Project Euler*.

O mundo SAP está, infelizmente, recheado de ABAPers que não fazem a mínima ideia do que estão fazendo. Não deixe que o despreparo dos outros seja a razão principal do seu triunfo.

5.4 O valor da revisão

A primeira vez que passei por um processo de *Peer Review* (revisão de pares), em que uma pessoa revisava absolutamente tudo que eu tinha gerado durante o desenvolvimento de um programa ABAP (incluindo documentação e testes), foi quando eu ainda era um estagiário.

Esse processo de revisão continuada manteve-se durante os 4 anos em que trabalhei naquela empresa. Sempre, antes de entregar um desenvolvimento para um cliente, algum programador mais experiente parava algumas horas

(já previstas no cronograma do projeto) e revisava tudo que eu havia feito.

Todas aquelas discussões que eu tive com programadores experientes sobre as técnicas de programação e arquitetura de software foram extremamente benéficas para a minha formação como ABAPer. Lembro-me de ficar constantemente frustrado quando alguém achava algum problema em um desenvolvimento meu, ao ponto de começar a fazer um Peer Review *antes* do Peer Review. Ou seja, eu mesmo revisava a coisa toda nos mínimos detalhes antes de entregar. Mesmo assim, os erros continuavam aparecendo. Erros menores, é verdade, mas ainda assim erros que poderiam atrapalhar o uso daquilo que eu havia criado.

Um fato importante a destacar é que essa mecânica funcionava muito bem, pois eu trabalhava em uma "Fábrica de Software" no início da minha carreira. Em vez de mandar os ABAPers até o cliente, todos trabalhavam remotamente em um só escritório, atendendo múltiplos projetos.

Existiam divisões de ABAPers para cada projeto, mas a fábrica dispunha de processos para aumentar sua produtividade, e um desenvolvedor poderia atender mais de um cliente se assim fosse necessário. Um dos indicadores chaves da fábrica era a qualidade do software entregue, portanto, era crucial que o trabalho fosse revisado no processo de Peer Review antes da liberação para o cliente (ainda mais por boa parte da equipe era composta de juniores e plenos, um dos pilares que geram dinheiro nesse esquema).

Pois imagine você qual foi o meu espanto quando eu saí dessa consultoria e da sua fábrica de software para trabalhar diretamente alocado em projetos de clientes. Quase nenhuma consultoria ABAP aplica algum processo de Peer Review para ABAPers em alocações desse tipo, tudo depende do cliente: se ele não tiver nenhum processo para manter a qualidade dos códigos, ninguém se importa.

Era extremamente estranho e assustador saber que um programa meu iria subir para um ambiente de produção, sem ter sido revisado. Tanto que eu fazia mais um Peer Review em cima do que eu mesmo desenvolvia, ou seja, o Peer Review antes do Peer Review (que era para ser antes do Peer Review, que, nesse caso, não existia).

Aos poucos fui notando que, além de ninguém se importar com revisões, muitos tinham um misto de medo e vergonha de que alguém olhasse os seus

códigos. Dependendo da pessoa, sugerir alguma melhoria técnica era assinar um atestado de discórdia eterna, pois ela iria te marcar como o cara **chato** que fica *fuçando* no programa dos outros. Não existia argumentação ou discussão em prol da melhoria mútua (afinal, a sugestão de melhoria poderia estar errada!), era somente uma birra boba, no maior estilo 4ª série. Surreal.

Ah, mas a coisa ficava ainda pior.

Acontece que eu sempre tive muito *coaching* no início de minha carreira. A cada final de projeto, fazíamos uma reunião de feedback, e o líder e programadores mais experientes destacavam pontos em que eu tinha ido bem e onde eu tinha cometido algum deslize.

Havia até uma pessoa dedicada a coletar feedbacks dos líderes do projeto onde eu havia passado, para então poder discutir comigo o direcionamento da minha carreira (em um programa de *mentoring*).

Voltando ao projeto direto no cliente, meu programa sem revisão que já estava em um ambiente produtivo, seja por conta das minhas autorrevisões ou por pura sorte, não havia causado nenhum tipo de problema. Após o tempo de suporte, o projeto chegava ao fim. E o que aconteceu? Exatamente *nada*, além da alocação no próximo cliente.

Feedbacks, processos de revisão de código, programação em pares e discussões técnicas deveriam ser algo **normal**, e não a exceção. Como é possível que, após mais de 30 anos de indústria SAP, as consultorias ainda não enxerguem os problemas que a falta desse tipo de prática causa para seus clientes?

Para começar, o ABAPer que nunca passou por nada disso cria um baita de um ego do tamanho do mundo. Se ninguém nunca reclamou do seu trabalho, e seus programas sempre foram para o ambiente produtivo e funcionaram (mesmo que depois de 1 zilhão de correções de erro), isso quer dizer que ele sabe programar muito bem, certo?

Não importa se futuramente aqueles que darão manutenção em seus programas passarem noites debugando um erro por conta da péssima estrutura do código. O ABAPer também quase nunca vê o usuário (mais sobre isso no capítulo 7), então também não importa se as telas com informação são tão confusas que, para extrair alguma informação de um relatório, seja preciso exportá-lo para o Excel (!!!) e tratar os dados, já que o programa não os trata por completo.

Como é possível elevar a qualidade dos seus profissionais, sem dizer quais são seus pontos fortes e fracos? As empresas que provêm serviços ABAP adoram achar que a solução para melhorar a qualidade da equipe é dar treinamentos, quando o que falta é uma boa dose de verdade. Ou seja, ser aberto aos seus funcionários quanto aos seus pontos fortes, fracos e sobre o que pode ser melhorado.

Onde você entra nessa história? É simples: peça para alguém mais experiente revisar seus códigos. Crie pequenas unidades de códigos automatizadas – uma classe, um conjunto de funções etc. – para serem utilizadas por mais de uma pessoa, e peça para que elas avaliem se a solução técnica atende às necessidades.

Dois programadores poderão resolver um mesmo problema de formas completamente distintas. Discutir as soluções e caminhos diferentes é um exercício riquíssimo. A cada final de projeto, peça para seu líder (ou até mesmo o cliente) uma análise do seu trabalho técnico. Veja se aquele modelo mental do "você profissional", criado pela sua cabeça, condiz com a realidade.

Mesmo sozinho em um projeto, existem inúmeras formas de conseguir feedback técnico. Uma maneira interessante de ganhar outras perspectivas em cima do seu trabalho é compartilhando-o pela internet.

Você pode fazer isso por meio de qualquer rede de compartilhamento de código, como o GitHub (que reconhece a sintaxe ABAP nos arquivos texto de final `.abap`), ou mesmo por meio de tópicos no fórum principal da comunidade, o SCN.

A prática de compartilhar código ABAP ainda não é popular no mundo SAP, mas está crescendo aos poucos. Dê uma olhada no filtro do GitHub para a palavra ABAP, disponível em http://bit.ly/2biqpmR, para ter uma ideia.

Se voc considerar que a SAP liberou somente em 2014 o compartilhamento de códigos ABAP em qualquer plataforma, até que 300+ repositórios envolvidos com ABAP, de alguma forma é um volume razoável (razões legais impediam o compartilhamento de código fora do site da SAP até meados de 2014. Sabe como é, empresas antigas são cheias desses rolos).

Independente do canal, é importante saber que receber feedbacks, técnicos ou não, é fundamental para a sua formação. Mesmo para um sênior que tenha crescido nesse cenário completamente estranho, sempre há tempo.

Para terminar, é essencial que você escolha bem quais conselhos dados durante algum tipo de revisão deverão ser incorporados no seu trabalho. Um bom revisor de código não vai impor o seu jeito de trabalhar. Exemplo de imposição: pedir para que você altere o símbolo de todas as expressões de igualdade de = para `EQ`. Isso não é nenhuma melhoria de código, é uma convenção que o revisor acredita ser "melhor".

A revisão que trará benefícios para a sua evolução como programador é aquela que questiona os motivos que levaram você a tomar as decisões técnicas que tomou, para implantar os requerimentos. Revisões de performance, melhorias de estrutura, dicas de comandos/técnicas que poderiam ter sido utilizadas é o tipo de revisão que trará valor.

Manias de código

Confesso que demorei algum tempo até entender que o meu jeito de programar não é a única forma de fazê-lo. Todo programação tem as suas manias, que não tem relação nenhuma com a estruturação do código, mas sim com a *forma* como ele é escrito.

Se levarmos em consideração a quantidade de palavras reservadas do ABAP, é fácil imaginar que cada programador (ou time) acaba criando o seu jeito de indentar e escrever os comandos. Acessando este site http://bit.ly/2byVRka, você poderá ver diversos ABAPers compartilhando suas manias.

Aproveite e compartilhe as suas também!

5.5 Estudos e comunidade

Já escutei inúmeras histórias de ABAPers mais antigos sobre como era difícil conseguir informação quando o sistema se popularizou no Brasil, no meio dos anos 90. Era comum abrirem uma apostila com o título "ABAP Básico" na frente de clientes, antes de começar a desenvolver um novo programa. Não haviam fóruns, a oferta de cursos era escassa (e muito cara!) e não existiam muitos consultores. O lema era aprender-fazendo, e seja o que os deuses quiserem.

Hoje em dia, todo programador aprende logo de cara que o Google é o seu maior aliado na busca de informações. Uma rápida pesquisa por algum comando ABAP retornará inúmeras páginas, com definições, explicações, exemplos etc. A informação é abundante, tanto que é preciso tomar um certo cuidado ao copiar todo e qualquer exemplo, já que muitos podem estar completamente errados.

O mais interessante é que nem todo esse conteúdo foi gerado pela própria SAP. Aliás, a maior parte foi gerado por outros ABAPers, que dispuseram do seu tempo para responder alguma dúvida, por vezes montando exemplos ou protótipos para exemplificar suas teorias.

O site da SAP Community Network (http://scn.sap.com) é o maior repositório de respostas a dúvidas, e abriga um número grande de profissionais dispostos a ajudar na solução de dúvidas, para os mais diversos problemas.

Quem dedica tempo respondendo às dúvidas de outras pessoas não são gênios da programação. São programadores como eu ou você, que passam pelos mesmos problemas, dúvidas e questionamentos. Já ouviu aquela história de que as pessoas que mais aprendem são aquelas que mais ensinam? "Gastar" tempo respondendo dúvidas de outros desenvolvedores é estudar.

A responsabilidade de responder alguma coisa relevante cria a necessidade de pesquisar sobre o tema e/ou criar protótipos, para não falar bobagem. E não é necessário muito tempo para esse tipo de atividade: uma pequena rotina diária de 20 minutos, lendo dúvidas de outros desenvolvedores e tentando ajudá-los, já é o suficiente.

Essa parte de dúvidas e respostas é a parte mais óbvia da interação da comunidade. Porém, há uma outra fonte ótima de leitura, que são os artigos criados por ABAPers sobre os mais diversos temas. Artigos interessantes geram discussões de altíssimo nível, que contribuem muito para o crescimento de todos os envolvidos.

Um bom exemplo são os artigos criados pelo Horst Keller, um dos responsáveis pela documentação do ABAP. Ele costuma criar diversos posts contendo as novidades das novas versões do ABAP, antes mesmo delas serem liberadas, com exemplos claros do "antes" e "depois" de cada um dos códigos. Como os clientes costumam demorar para atualizar seus sistemas, muitos desses textos têm uma relevância garantida por muitos meses, como o

artigo *ABAP Language News for Release 7.40 – Inline Declarations* (http://bit.ly/2bqbUip) , criado em 2013, porém relevante até os dias atuais. Como o ciclo de atualização do ABAP nas empresas é enorme, esse artigo continuará relevante por um bom tempo.

A visão de que estudar é somente ler um livro ou fazer um curso está completamente equivocada. Envolver-se com a comunidade é estudar.

Um dos posts mais clássicos do SCN é uma discussão gigantesca sobre um dos temas mais polêmicos da performance em programas ABAP: o uso do `FOR ALL ENTRIES` versus o uso do `INNER JOIN` para "unir" tabelas na hora de realizar seleções (http://bit.ly/2bkOfxG) . Esse tópico tenta encerrar a discussão dizendo que os `JOIN`s serão melhores que os `FOR ALL ENTRIES`, na maioria dos casos. Lá há diversos comentários, links suportando as teorias e mais de 50 mil visualizações. Mesmo assim, aposto que você encontrará algum ABAPer que acredita que `FOR ALL ENTRIES` são **sempre** melhores "porque sim", facilmente. Experimente perguntando para alguns.

Estudar pelo envolvimento com comunidade é tão ou mais importante do que pagar cursos oficiais (ou secundários) sobre alguma técnica específica. As discussões e níveis de detalhes de uma comunidade engajada pode atingir níveis dificilmente cobertos por um curso de poucas horas.

Minha última recomendação de estudos é a de que você gaste tempo aprendendo absolutamente qualquer tecnologia que não tenha **nada** a ver com SAP. Os benefícios dessa prática são infinitos. Por mais que existam diversos guias do mundo SAP que explicam Orientação a Objetos no ABAP, passei a entender mais a fundo as decisões do time que desenhou o ABAP Objects, quando decidi aprender C++ (que tem uma, adivinhe, *comunidade* que disponibiliza vários guias gratuitos muito bons).

Todas as linguagens e técnicas de programação evoluem em conjunto, e muita coisa que foi criada para uma determinada tecnologia pode ser replicada em outra. Aprofundar seus conhecimentos em ABAP é mandatório, mas ficar só nisso acaba limitando a sua capacidade.

5.6 ABAP é um lixo, vamos vender coco na praia

Que atire a primeira `BADI` quem nunca escutou a frase: "Trabalhar com ABAP é muito ruim, vamos largar tudo e vender coco na praia". É uma constante no meio SAP escutar pessoas bradando que estão infelizes em suas posições atuais e preferiam estar fazendo algo diferente (eu mesmo já reclamei com pessoas mais próximas sobre isso).

Entretanto, tenho um pensamento que vai um pouco contra dessa lenda urbana de que abrir cocos é melhor que debugar a VA01 (nome popular da transação do programa de ordens de venda). Para que você entenda minha linha de raciocínio, vamos explorar um pouco como nasce esse sentimento de que "está tudo zoado".

Como surge a desmotivação generalizada

- Projetos com prazos apertados;
- Gerentes/líderes tapados;
- Time comercial que vende errado;
- Equipe técnica ruim;
- Tecnologia datada e antiga;
- Mãos atadas às decisões dos clientes;
- Precariedade das práticas;
- Padrões e convenções arcaicas;
- Uma infinidade de horas extras;
- Usuários malditos e funcionais odiosos;
- ABAPers pilantras;
- BASIS que trabalham sem vontade;
- Incapacidade de usar tecnologias novas;

- Recrutadores picaretas;
- SAP é uma empresa que só atrapalha;
- O standard é horrível;
- <insira sua reclamação aqui>.

Qualquer pessoa com alguns anos na área ABAP fica desmotivado quanto pensa no futuro, em grande parte por uma ou várias das razões anteriores. Normalmente, você fica extremamente empolgado com um mar de possibilidades iniciais (júnior), para então tomar algumas "porradas" da vida ABAPer em projetos complicados que começam a lhe deixar irritado (pleno), finalmente terminando em uma posição de conformidade com os problemas por conta de salários bons para a realidade brasileira (sênior). Boa parte da discussão sobre salário e taxas vem desse conformismo.

No mundo SAP, somos afogados anualmente em um mar de novos nomes, siglas e produtos pela comunidade SAP mundial: HANA, Fiori, Gateway, UI5, blá-blá-blá. As mesmas indagações que já discutimos se repetem: o que estudar? O que fazer? Para onde eu vou? ABAP é um lixo então, é isso?

Esse medo aumenta ainda mais uma desmotivação generalizada pelo entendimento de que aquilo com que você trabalha não é o suprassumo tecnológico da empresa que provê a ferramenta para seu trabalho (a SAP). É como se alguém falasse: "Olhe que legal esse teletransportador que faz você ir do Brasil até a China em 5 segundos! Agora cale a boca e vá a pé".

Saindo do mundo SAP e olhando para os lados, encontramos um mundo que é, aparentemente, maravilhoso. Startups divertidas, trabalho remoto, tecnologia de ponta, pessoas felizes, capacidade real de aprendizado, eventos descolados, equipes qualificadas, <insira algo que você não tem e gostaria hoje aqui>. Todo mundo sabe do ditado popular: "A grama do vizinho é mais verde". Em um mundo onde você trabalha em uma sala escura, sem água, rede e em condições precárias, não dá para piorar, não é mesmo?

Problemas, problemas, problemas...

O futuro não é promissor?

Pense em tudo que você acabou de ler. Eles são problemas específicos do mundo SAP, ou estão relacionados com várias áreas? Esse monte de coisas está presente em tudo quanto que é lugar, e a maior parte delas foge do nosso controle.

Nem mesmo uma profissão perfeita está livre de problemas. Vamos pegar o exemplo de alguém que vende cocos na praia. A pessoa acorda de manhã, passa um protetor solar, arranja os cocos, empurra seu carrinho até a praia, senta olhando para o mar, puxa uma câmera profissional, tiras diversas fotos e posta na Instagram/Facebook para massagear seu ego e mostrar para o mundo como a sua vida é ótima. Entre análises visuais de 834 ondas por hora na beira do mar, a grana alta aparece de forma mágica em sua conta permitindo que a pessoa volte para a casa após um dia superprodutivo, guarde seus cocos em um lugar seguro e vá jogar um videogame, beber cerveja ou <insira algo que você goste aqui>.

Após 1 ano nessa vida, se alguém perguntasse para a pessoa "como anda o mercado dos cocos", o que você acha que ela responderia? Eu **aposto** que boa parte das pessoas em situações similares responderia "não está fácil". Reclamariam pelo puro prazer mórbido de reclamar.

O grande vilão do seu futuro é você mesmo. Os problemas do mundo terão impactos inferiores aos problemas trazidos por fazer mal a si mesmo.

Reclamar por reclamar é um mecanismo de defesa para demonstrar aos outros que nós temos uma capacidade superior àquela que utilizamos, para fazer o que fazemos no nosso dia a dia. Eu cansei de reclamar por reclamar na minha vida, para 10 minutos depois continuar fazendo exatamente a mesma coisa de que reclamei (e não me orgulho disso nem um pouco).

Eu ainda faço isso, não é fácil mudar. Isso é um costume, um vício, uma cultura. Algo que é extremamente comum no mundo SAP e em outras áreas, algo que é feito milhares de vezes diariamente em todos os projetos, clientes e consultorias.

Se refletirmos em volta desse monte de reclamações e problemas, veremos que todos realmente existem, sempre existiram e sempre existirão. É você que decide como lidar com eles e o quanto isso tudo o afetará.

Vá treinar

Reclamar constantemente de que tudo está um lixo não leva a absolutamente nada. Nós gostamos de criticar a SAP por absolutamente tudo o que acontece, como se ela fosse responsável por fazer as coisas direito, para que nós possamos triunfar da forma certa. A SAP faz o que ela entenda o que é melhor para ela, assim como você deveria fazer o que acha melhor para você em vez de reclamar, reclamar, reclamar e não fazer nada.

Meu mestre de Hapkido costumava dizer uma coisa que ficou muito marcada em minha vida: **vá treinar**. Em vez de ficar discutindo se a arte marcial X é melhor que a Y, vá treinar. Não perca tempo reclamando das federações, vá treinar. Não alimente o ego querendo provar que você é o mais forte, vá treinar. Não seja vencido pela preguiça, vá treinar.

Não fique sonhando com a venda de cocos na praia, vá debugar.

Quer aprender coisas novas no SAP? Vá em frente, esforce-se! Quer sair do mundo SAP, então mexa-se! Estude algo diferente, crie projetos novos, corra atrás do seu sonho maluco. Não quer estudar nada, não quer aprender nada novo, quer continuar com o seu ABAP e pronto? Ótimo, melhore seu nível ABAP e não se sinta mal com a escolha.

Não seja contagiado por visões pessimistas. Isso não vai levar a nada. Você não precisa concordar com tudo que acontece em sua volta, mas deixar de fazer algo sobre aquilo que lhe incomoda é onde mora o perigo. Enquanto você ficar parado, nada acontece. O mundo muda, mas você fica na mesma.

E aí, o que vai ser?

Capítulo 6

Os tabus do mundo ABAP

Conforme você for (digi)evoluindo como ABAPer, você entenderá que boa parte do trabalho é pegar um monte de código antigo e tentar fazer algo com aquilo. Por serem códigos monstruosos, fica muito difícil entender toda a lógica para poder criar uma nova alteração que condiga com a estrutura originalmente preparada para o programa.

Na prática, cada novo desenvolvedor vai incorporar as novas alterações do seu jeito, torcendo para que suas inclusões não quebrem nada que já funciona. E aqueles que ajustaram o código podem ser pessoas de níveis técnicos diferentes, com seus próprios costumes e formas de trabalho.

Esses programas tornam-se grandes colchas de retalhos. Diversas lógicas espalhadas, cada uma à sua maneira, gerando titãs de 30 mil linhas. Só que, apesar desse monte de códigos malucos e complicados, o objetivo final do programa provavelmente **continua o mesmo**. Ele pode fazer uma ou duas

coisas a mais, entretanto deve continuar entregando o resultado final para que foi originalmente criado.

A parte mais legal dessa história é que pouquíssimas pessoas serão capazes de dizer o que o programa faz por dentro (talvez ninguém saiba). **Funciona** e isso é o suficiente.

Um programador consegue aprender bastante criando, mas também consegue aprender muito lendo códigos alheios. O programa monstro funciona, então tem o seu valor. Um ABAPer júnior pode ler as 67 mil linhas de código e evoluir muito como desenvolvedor, não é mesmo? ABAPers experientes responderiam que "ele pode aprender, mas precisa tomar **muito** cuidado". Sendo um programa antigo, deve estar inundado de técnicas ultrapassadas e estruturações ineficientes, servindo como histórico da linguagem, mas não como base do que deve ser usado na criação de um novo programa.

Essa breve fábula de um programa jurássico repete-se ao analisarmos a prática ABAP como um todo. Pense bem: no passado, alguém criou uma forma de trabalho, visando consistência e qualidade no processo de desenvolvimento. Ao longo dos anos, muitas pessoas foram contribuindo para que isso evoluísse. Será que todas elas avaliaram o processo completo de forma objetiva para implantar "melhorias"? Ou será que cada uma também estava munida de seus próprios costumes e formas de trabalho, implantando determinada nova prática como se fosse a mais correta?

O processo de desenvolvimento ABAP também pode ter virado uma colcha de retalhos? Existem práticas sem sentido que são seguidas por equipes de projeto, sem questionamentos?

Vamos ver alguns aspectos cruciais da prática ABAP para validar se alguém realmente avaliou o processo como um todo, ou se simplesmente adicionou o seu `PERFORM resolve_somente_o_meu_problema` para entregar a demanda.

6.1 Fantasmas do passado

Sendo o ABAP uma linguagem de programação de certa idade, é natural imaginarmos que o mercado já deve ter passado por diversas fases diferentes. Tudo o que gira em torno do mercado de software sofreu diversas mudan-

ças nos últimos 30 anos (época em que o ABAP foi criado), e essas alterações certamente atingiram a prática ABAP nas empresas.

Eu gosto de dizer que "no mundo corporativo a roda gira muito mais devagar do que gostaríamos". Somos diariamente inundados em redes sociais e sites de notícia por maneiras inovadoras de trabalho, times espalhados em vários lugares do globo, equipes altamente produtivas implantando metodologias malucas e programadores capazes de criar sistemas para resolver os mais diversos problemas.

Esse dinamismo não está presente em ambientes corporativos, por um motivo óbvio: qualquer alteração brusca nos times internos de uma empresa tem a possibilidade de gerar impactos negativos em seus negócios. Mudanças costumam ser meticulosamente analisadas antes de serem incorporadas pelos times de desenvolvimento de grandes empresas.

Na prática, profissionais SAP estão acostumados a enfrentar práticas antiquadas de trabalho. As formas antigas vão desde metodologias de projeto até a programação nossa de cada dia. Muitos fantasmas do passado assombram a vida dos ABAPers, que muitas vezes incorporam esses fantasmas sem perceberem.

Para exemplificar esse processo, no que diz respeito aos códigos, um dos fatores que mais contribuem para esse fenômeno é o qual veremos na sequência.

A retrocompatibilidade

O SAP ERP mantém a retrocompatibilidade com versões mais antigas do ABAP. Isso quer dizer que alguma funcionalidade antiga continua válida em programas criados hoje, mesmo que já exista um forma mais nova e robusta de implementá-la. Conforme explicado no capítulo 1, o SAP contém seus próprio programas escritos em ABAP, e a quantidade de linhas de código *standard* no sistema é realmente grande. São milhares e milhares de programas criados para as mais diversas funcionalidades e que continuam sendo usados em ambientes produtivos nos dias atuais.

Normalmente, a reação ao nos depararmos com algum código *standard* é a de jogar tudo fora e reescrever completamente a aplicação. Porém, seria uma loucura total tentar refazer o SAP ERP do zero – isso poderia matar

completamente o produto.

Joel Spolsky nos conta um pouco do que aconteceu com o Netscape quando o seu time de desenvolvimento tentou recriar o produto, no clássico post *Things You Should Never Do, Part I* (http://bit.ly/2bkOfxG) . A situação aqui é semelhante: por mais que os códigos não sejam perfeitos e usem técnicas datadas, eles já passaram por diversos níveis de aprovações de utilização em ambientes produtivos. Para a estratégia da SAP como empresa, manter a retrocompatibilidade do ERP é algo crucial para seu negócio dar certo.

6.2 Impactos nos processo de desenvolvimento

Voltando à questão da inovação, se as empresas demoram para incorporá-las, imagine seus profissionais. A programação ABAP é inundada por mitos, teorias e manias que já caíram por terra há muito tempo. A retrocompatibilidade contribui para que cada ABAPer crie seu próprio mundinho da programação, recheado das coisas que **ele** sabe que funcionam. O que é novo, diferente ou estranho deve ser ignorado, até que algum cliente exija. Essa abordagem tem reflexo em desenvolvedores, funcionais, líderes, gerentes... É tentador acomodar-se em meio àquilo que você já conhece (o seu mundinho), mas sair constantemente da sua zona de conforto é essencial para manter uma carreira de sucesso.

Muitos zumbis nascem ao incorporarem práticas sem questionar se elas fazem ou não sentido. Todos eles compartilham uma mesma condição, que eu gosto de chamar de "síndrome do robô".

Fuja da síndrome do robô

Uma construção comum no mundo do ABAP:

```
LOOP AT table INTO line.
  CALL FUNCTION 'BUSCA_INFORMACOES_ADICIONAIS'
    EXPORTING
     material = line-material
    TABLES
     details = material_details[].
```

```
 (...)
ENDLOOP
```

Simples, não é? Considere que a função `BUSCA_INFORMACOES_ADICIONAIS` é uma função standard que retornará todas as informações adicionais dos materiais para a tabela `MATERIAL_DETAILS`. É ótimo descobrir a existência de algo mágico que retorna todos os dados formatados exatamente do jeito que precisamos. E é nesse sentimento de felicidade onde mora o perigo.

Como praticamente todo o código ABAP criado pela SAP é aberto para visualização e debug (é muito raro encontrar algum código que não seja possível visualizar), constantemente tentamos resolver nossos problemas usando as mesmas funções que os programas standard utilizam. Aliás, essa é uma boa prática: em vez de reescrever a roda, devemos sempre procurar por funções ou classes standard que resolvam nossos problemas, principalmente por conta da complexidade envolvida na manipulação de dados e regras de negócio dentro do SAP.

Só que nem sempre o ABAPer da SAP criou determinada função pensando no reúso. Muitas vezes ela foi criada pensando no contexto daquele programa standard, ou em um grupo de funções específico. Ela certamente resolve os problemas gerados por aquele programa, mas pode não ter sido criada nem otimizada pensando na reutilização em outros códigos.

A função do exemplo inicial foi criada para buscar os dados de um material e, aparentemente, resolve todos os problemas. Só que esse tipo de faz-tudo pode fazer milhões (ou melhor, bilhões ou trilhões) de seleções e acessos ao banco de dados, destruindo a performance do programa (ou até do sistema).

Ao longo desses anos trabalhando com ABAP, tenho notado que poucas pessoas preocupam-se em analisar o código dentro da função antes de utilizá-la dentro de laços para milhões de registros. Ludibriado pela facilidade com que os dados chegam *mastigados* ao programa, é fácil dar um tiro no seu pé ou no de alguém que algum dia vai ter de fazer uma análise de performance no tal programa, que deve rodar mais lento que o trânsito de São Paulo em um dia de chuva.

Como resolver?

Infelizmente, não há uma regra que pode ser aplicada sempre: cada caso é um caso. O ideal é que você verifique o que a função faz antes de sair enfiando-a em tudo quanto é `LOOP` do seu código. O standard pode ter códigos realmente complexos, mas utilizando transações como a ST05 ou SE30, é possível ter uma rápida ideia se algumas das funções podem ser prejudiciais.

Vou contar uma pequena história de exemplo. Certa vez, ajudei em uma análise de performance em um programa gigantesco, e o maior problema era uma chamada de função para buscar a lista técnica dos materiais, executada diversas vezes sem motivo:

```
LOOP AT purchase_orders INTO purchase_order.
  CALL FUNCTION 'CS_BOM_EXPLOSION'
    EXPORTING
      matnr = purchase_order-matnr
    TABLES
      (tabelas de retorno c/ a lista técnica).
ENDLOOP.
```

Essa função executa muitos `SELECTs` em seu processamento, e não era necessário chamá-la tantas vezes. A tabela `PURCHASE_ORDERS` contém itens de documentos de MM (*material management*, um dos módulos do SAP), e itens de diferentes documentos podem ter o **mesmo** material. A lista técnica de um material não é um dado que sofre alterações com frequência, portanto, não existe nenhuma necessidade de executar essa função duas vezes para o mesmo material dentro de um `LOOP`. Uma solução melhor é montar um cache interno para as chamadas dessa função.

No caso que ajudei a resolver, fizemos um controle muito simples para que a função não fosse executada duas vezes para o mesmo código de material, melhorando a performance do programa em cerca de 70%. A solução utópica seria refazer as seleções dentro do programa, recriando acessos únicos a cada uma das tabelas no banco.

Nas poucas horas que tínhamos para refatorar o programa, essa solução seria completamente inviável, pois listas técnicas contêm regras de negócio muito complexas, e muitas variações a serem consideradas. A função realmente facilita por conter a lógica standard que considera configurações, mas

é preciso uma análise para entender um pouco melhor como ela funciona antes de sair utilizando-a de qualquer jeito.

Esse exemplo de análise antes do uso é aplicável em diversas situações do mundo ABAP. Ele também nos revela que existe um problema muito maior por baixo de uma singela função usada de forma errada.

A síndrome do robô

Normalmente, um robô não é criado para pensar: ele deve executar uma tarefa pré-programada de forma eficiente, no menor tempo possível. ABAPer nenhum gosta de ser enxergado como um *robozinho* que bebe café e o converte em linhas de código. Todos também revelam um certo desconforto quando alguém brinca que a divisão ABAP de um projeto tende a virar uma pastelaria conforme os prazos ficam apertados ("agora me vê um `REPORT` de queijo, agora uma função de escarola").

O fato irônico é que muitos se comportam como robôs quando criam soluções a partir de cópias de trechos de código, sem fazerem a menor questão de entender o que está por trás daquela *classe mágica* ou *função faz-tudo*.

Você pode estar pensando: "Ah, mas é impossível entender tudo o que uma função/classe standard faz", e eu digo que códigos standard são realmente complexos de serem compreendidos. Entretanto, entender ao menos a base do funcionamento do trecho que você estiver reutilizando torna-se essencial para criar uma aplicação robusta. Saber que a função `CS_BOM_EXPLOSION` faz seleções recursivas salva drasticamente a performance do programa que a usa, e o mesmo é válido para outras funções, classes, ou praticamente qualquer código copiado do standard (e fóruns de internet).

Mais um exemplo da síndrome são as pessoas que têm um prazer mórbido pela utilização de `CLEAR`, `REFRESH` e `FREE1`. São relatórios, *forms* e métodos Z que começam com a limpeza de variáveis que **acabaram** de ser declaradas. O sujeito teve um problema de variáveis *sujas* uma vez na vida, e prefere limpar tudo por precaução em vez de entender o que ocasionou o erro da primeira vez. Nessas situações, nascem os robôs.

A dinâmica do SAP e da evolução da linguagem faz com que seja necessário reavaliarmos aquilo que se popularizou como *a forma correta*, diariamente. Há algo no seu dia a dia ABAPer que se encaixa na síndrome do robô?

Quanto tempo você gastaria para entender um pouco do funcionamento *daquela* coisa que você utiliza e sempre salva a sua vida?

Tente achar respostas para essas questões. A realidade dos projetos certamente obrigará você a usar coisas sem saber completamente como elas funcionam, mas tenha sempre em mente que, mesmo copiando uma solução de algum outro lugar, a responsabilidade da cópia recém-criada será sempre 100% **sua**.

A síndrome do robô vai muito além da codificação. Pense nas documentações, no controle de versões, na padronização e nos processos de revisão. Não se contente simplesmente com algo que *só* funciona. Tente entender aos poucos o que realmente acontece por trás de cada um dos códigos e processos que você utiliza.

Não entre no modo automático. Fuja da síndrome do robô aproveitando as dicas a seguir!

6.3 Nomeclaturas em ABAP

Se existe uma pessoa que certamente não fugiu da síndrome do robô, foi aquela que disse que o padrão de nomenclatura `<fs_>` servia para alguma coisa. Responda rapidamente: qual outro tipo de varíavel no mundo ABAP que usa `<>`, sem ser os `FIELD-SYMBOLS`? Nenhuma? Certo! Então, por que utilizar o desnecessário `fs_`? O identificador `<>` não é o suficiente?

O motivo disso é simples: "tudo vem da padronização". Não basta ter a palavra `PERFORM` ou `FORM` para identificar que aquilo é uma chamada de um bloco de código, é preciso colocar um inútil `F_`. Não basta que um método possua formas bem específicas de ser disparado, é preciso colocar `M_` em seu nome. Isso tudo é realmente necessário?

Sei muito bem que as grandes empresas possuem os seus próprios padrões de nomenclatura e forçam a utilização. Para o pobre ABAPer alocado, não sobre muita escolha: você precisa seguir os padrões que o cliente decidiu; caso contrário, alguém fará com que você "ajuste" seu programa inteiro para que ele possa seguir para a produção.

Essa história de colocar um identificador do tipo `V_` antes de variáveis não nasceu no mundo SAP. Isso é uma prática muito conhecida no mundo

do desenvolvimento de software, chamada **notação húngara**. A ideia básica é colocar um identificador antes do nome da variável, para que você possa saber de que tipo é a variável sem precisar navegar até a sua definição.

É assim que você poderia saber, por exemplo, que `sNome` é uma `String`, só de olhar para a variável. O seu uso é fruto de muita controvérsia e discussões constantes. No artigo em inglês da notação húngara do Wikipédia, podemos ver citações de desenvolvedores famosos, alguns a favor e outros contra (http://bit.ly/2biqMOh) . Sua adoção/aversão não é unânime.

De volta ao ABAP, a notação húngara certamente ajudou diversos desenvolvedores da época do `HEADER LINE` (forma de construir tabelas que está obsoleta). Vamos declarar uma tabela interna com `HEADER LINE`:

```
DATA: mara TYPE TABLE OF mara WITH HEADER LINE.

mara-matnr = '1234'
```

Se precisássemos de uma estrutura com dados da `mara`, poderíamos declarar da seguinte forma:

```
DATA: mara TYPE mara.

mara-matnr = '1234'
```

E se o nosso programa tivesse de ter, ao mesmo tempo, um mesmo bloco de código, a `mara` tabela interna e a `mara` como estrutura, como identificar qual é qual? A solução mais prática adotada na época foi a notação húngara, usando um identificador antes do nome de cada uma das variáveis:

```
DATA: t_mara  TYPE TABLE OF mara WITH HEADER LINE,
      wa_mara TYPE mara.

t_mara  = '1234'.
wa_mara = '1234'
```

Dessa forma, fica mais fácil saber qual variável é qual. Assim foi naquela época e ainda é nos dias atuais. A notação húngara ainda é o padrão dominante para diversos projetos ABAP.

Bom, se temos uma ótima forma de identificar o tipo da variável, para que perder tempo explicando sua utilidade, não é mesmo?

```
DATA: v_tabix  TYPE sy-tabix,
      v_tabix2 TYPE sy-tabix.
```

Ah, são todas variáveis simples, e tem o `V_` ali, o que é mais do que o suficiente! Será mesmo? Em algum momento, o desenvolvedor preocupou-se em explicar para que serve aquela variável? Vamos tentar reescrever o mesmo código de outra forma:

```
DATA: indice_tabela_materiais TYPE sy-tabix,
      indice_tabela_clientes  TYPE sy-tabix.
```

Em qual dos dois exemplos ficou mais claro o motivo de existência de cada uma das variáveis?

Na minha visão, o maior problema dos identificadores das variáveis é que elas fazem com que o ABAPer vá pelo caminho mais fácil: se o nome já tem o identificador do tipo (`T_`, `V_` etc.), não é preciso se preocupar em dar um nome que explica a utilidade daquela área de memória.

Existe ainda um sentimento generalizado de que as variáveis devem ter um nome pequeno. No passado, manter variáveis de nomes curtos fazia sentido, tanto pela limitação de 80 colunas do editor ABAP quanto pela dificuldade de visualizar códigos grandes no debug antigo:

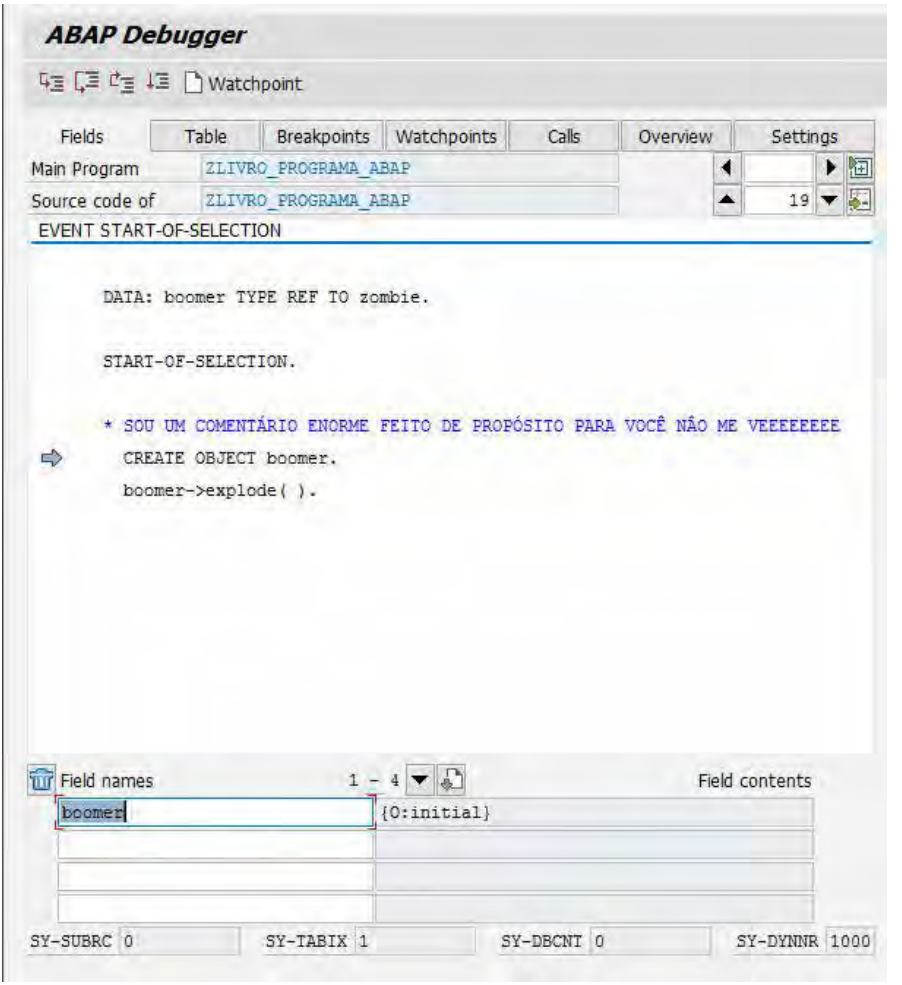


Fig. 6.1: Isso acontecia direto com programas complexos e era extremamente irritante

Desde 2006, o editor e debug do ABAP evoluíram e não possuem essas limitações. Outra coisa que contribuiu muito para a popularização das variáveis com nomes pequenos e ininteligíveis foi o dicionário ABAP. `MATNR` é material, `BUKRS` é empresa e `GJAHR` é o ano fiscal. Mas a sua variável não precisa chamar `V_MAT_GJ`.

Nomenclaturas como `<fs_>` e `f_` não servem para absolutamente nada, mas mais importante que isso é descrever corretamente o que virá **depois** dos identificadores. Não há necessidade de manter uma variável com o nome pequeno sem motivo. Quanto mais descritiva ela for, melhor. Mesmo que você seja obrigado a usar um desses padrões, não existe razão para fazer disso uma muleta e deixar de descrever o real propósito de suas variáveis.

Da próxima vez que você for criar uma variável chamada `V_AUX`, pense bem no motivo que levou à sua criação. Eu tenho certeza absoluta de que, ao responder a pergunta "para que serve esta variável?", você automaticamente vai descobrir um bom nome para ela.

6.4 Vida e obra das constantes inúteis

Por falar em variáveis, você já viu algum código com a seguinte declaração?

```
CONSTANTS: c_x TYPE c value 'X'.
```

Eu aposto que sim. Aliás, eu apostaria que você já fez, faz ou vai fazer isso em algum momento da sua carreira, pois a síndrome do robô tem uma penetração nos padrões de empresas muito maior do que podemos imaginar. Mas não colocaremos o `LOOP FROM sy-tabix` antes do `READ TABLE BINARY SEARCH`, vamos entender um pouco por que esse tipo de constante é completamente inútil.

O que é uma constante?

O comando `CONSTANTS` é usado na declaração de constantes (sério?), ou seja, variáveis com um valor que nunca mudará em tempo de execução:

```
CONSTANTS tipo_documento TYPE char4 VALUE 'WXYZ' .
```

Você pode usar a variável `TIPO_DOCUMENTO`, tendo a certeza de que seu valor nunca será alterado após o início do programa. Você pode até criar uma estrutura constante, veja:

```
CONSTANTS:
BEGIN OF estrutura,
```

```
  inteiro  TYPE i VALUE '123',
  char     TYPE c VALUE 'A',
  numchar  TYPE n VALUE '2',
END OF estrutura.

WRITE estrutura-inteiro.
```

Essa mecânica de funcionamento é bem simples de ser assimilada. Só que a parte mais importante está na resposta para a seguinte pergunta: quando devo utilizar uma constante?

Imagine que você precisa fazer um programa que mostrará dados somente de uma determinada empresa e o código da empresa será utilizado em diversas seleções e validações dentro do código. Se você usar diretamente no código o valor da empresa e algum dia ele mudar, todo o código-fonte precisará ser alterado por conta de uma simples troca de valores.

Esse é o famoso *hardcode*, um valor escrito explicitamente sem nenhum tipo de interação com o usuário ou fontes externas (tabelas de parâmetro, arquivos etc.), também chamado de *marreta* em alguns casos.

```
material-werks = 'ZMB1'.

IF material-werks = 'ZMB1'.
* Faça algo
ENDIF.

CASE material-werks.
  WHEN 'ZMB1'.
*  Faça outro algo
ENDCASE.
```

Se o valor da empresa (`WERKS`) precisasse mudar, você teria que alterar três diferentes pontos do programa. Em vez de espalhar o valor `ZMB1` por todos os lados, é melhor criar uma constante. Assim, você garante que somente um lugar precisará ser ajustado.

```
CONSTANTS: empresa TYPE marc-werks VALUE 'ZMB1'.

material-werks = empresa.
```

```
IF material-werks = empresa.
ENDIF.

CASE material-werks.
  WHEN empresa.
ENDCASE.
```

Tudo bonito, tudo lindo, tudo perfeito. "Isso tudo é muito óbvio", você pode pensar. Então, responda rápido: por qual motivo existem quilos, litros e toneladas de constantes com `C_X` espalhadas pelos códigos?

Hardcode e a banalização das constantes

Quem falou que programadores deveriam evitar hardcodes foi sábio. Porém, em momento algum ele disse que **todo valor escrito de forma literal no código é uma constante**.

Cansei de ver constantes `C_X = 'X'` espalhadas por programas ABAP. Isso é totalmente desnecessário, o código fica horrível de se ler e o valor da variável **nunca** poderá ser alterado, já que tem significados **diferentes** em cada um dos lugares onde está sendo usado. Exemplo dessa coisa horrível:

```
PARAMETERS:
  p_docto AS CHECKBOX,
  p_users AS CHECKBOX,
  p_soma AS CHECKBOX.

CONSTANTS: c_x TYPE c VALUE 'X'.

IF p_docto = c_x.
* Algo com o documento
ENDIF.

IF p_users = c_x.
* Algo com o usuário
ENDIF.

IF p_soma <> c_x.
```

```
* Some alguma coisa
ENDIF.
```

Se você mudar o valor de `C_X` para `SPACE`, isso não quer dizer que todos os lugares onde `C_X` está sendo usado têm de ser alterados. Essa é uma variável que não precisaria ter sido criada, pois **nunca** será alterada, servindo somente para deixar o código irritantemente mais chato de ser lido.

A linguagem ABAP não possui valores booleanos (*true* ou *false*), portanto, o `'X'` é utilizado em vários lugares para descobrir se um campo está "ligado" (*true*, `'X'`) ou desligado (*false*, `SPACE`).

Reescrevendo o código anterior, imaginando que exista o suporte a valores booleanos, fica ainda mais claro que não faz o menor sentido existir essa constante:

```
* Imaginando que ABAP tenha booleanos:

IF p_docto IS true.
*  Algo com o documento
ENDIF.

IF p_users IS true.
*  Algo com o usuário
ENDIF.

IF p_soma IS false.
*  Some alguma coisa
ENDIF.
```

Simulando Booleanos no ABAP

O ABAP não contém suporte a booleanos, mas a SAP disponibiliza uma forma de "simular" seu uso em seu código. Por meio do `TYPE-POOLS: abap`, você consegue ter acesso ao tipo `abap_bool`, além de constantes `abap_true` e `abap_false`. Nenhuma dessas constantes e tipos são **reais** booleanos, são constantes do tipo `C`, que podem ter os valores `'X'` ou `SPACE`. A documentação do ABAP contém guidelines de como aplicar corretamente essas constantes no seu código: http://bit.ly/2biqMOh.

Pode parecer muito simples colocar somente uma constante burra de exemplo, mas é muito fácil encontrar `REPORT`s com mais de **20** constantes que não precisavam estar ali, deixando o código aparentemente mais confuso do que deveria.

Passei por projetos que obrigavam os desenvolvedores a criarem constantes para todos os valores literais existentes nos códigos. Nessas situações surgem soluções ainda mais contraprodutivas para constantes, como o "maravilindo" `C_SPACE`:

```
CONSTANTS:
 c_x     TYPE c VALUE 'X',
 c_space TYPE c VALUE ' '.

IF p_docto = c_x.
*  Algo com o documento
ENDIF.

IF p_users = c_space.
*  Algo com o usuário
ENDIF.
```

Estranho, não é? E o martírio não para por aí. Colocar o nome da constante igual ao seu valor é assinar um contrato de "problema a longo prazo" dentro do código.

```
CONSTANTS c_zmb1 TYPE CHAR4 VALUE 'ZMB1'.
```

Se um dia o valor `'ZMB1'` mudar para `'ZMB2'`, ficará *muito* mais fácil entender que o código usa a `C_ZMB1` em um `IF`, quando, na verdade, seu valor é `'ZMB2'` (é claro que não).

É exatamente por isso que eu nunca confio em constantes que contêm nomes parecidos com valores. Depois de um tempo trabalhando com SAP, dá para sacar que uma `C_2200` é o número de um centro ou planta, só não dá para ter certeza de que o seu valor é realmente `'2200'`. Podemos apelidar cada uma dessas constantes de *abomináveis constantes das neves*: "abomináveis", porque elas realmente são abomináveis; "das neves", porque soa divertido.

Mas enfim, quando eu preciso usar uma constante?

Vale o bom senso e a correta análise da regra de negócio. Se existe um valor que será usado em diversos pontos do programa, daqueles que o cliente/funcional não deverá nunca mudar sem requisitar uma alteração na especificação e que importam para o processo do desenvolvimento (tipo de documento, uma empresa, um número de conta contábil etc.), você pode considerar utilizar uma constante.

O melhor mesmo é nunca deixar o valor cravado no programa, tentando sugerir o uso de uma tabela de parâmetros, como a `TVARV` (acessando a transação `STVARV` para manipular). Isso garante que o cliente não precisará contratar um ABAPer somente para alterar um pequeno valor, deixando seu programa configurável.

Deixo registrado que, em todos esses anos nessa industrial vital, este é o tópico que mais revela as amarras impostas pelos processos jurássicos de qualidade de código, aplicados em algumas empresas. Essas travas, aliadas a desenvolvedores que preferem executar trabalhos como robôs em vez de ponderarem sobre o que fazem, atrapalham constantemente a qualidade dos programas gerados em projetos.

Abaixar a cabeça para uma metodologia e/ou processo quando ele claramente não faz sentido onde está sendo aplicado é negligenciar a possibilidade de melhorias no ciclo de desenvolvimento de softwares e projetos. Tome cui-

dado para não entrar no modo automático, e não deixe que processos de revisões arcaicos e *guidelines* de desenvolvimento datadas onerem a qualidade do seu trabalho.

Para terminar, é claro que eu consigo imaginar um cenário no qual algum processo de QAS (*quality assurance*, ou garantia da qualidade) sem noção vai obrigá-lo a emburrecer o seu código. Minha dica é: apresente argumentos válidos contra a prática que não faz sentido. Se você conseguir mostrar que aquela correção que está sendo imposta vai piorar o código, há grandes chances de a própria equipe de QAS reconsiderar o *guideline* burrico.

6.5 Pare de escrever documentações que não serão lidas

Na área de TI, quase tudo se resume ao tempo que você, desenvolvedor, vai gastar para fazer X atividades. Existem milhares de técnicas e maneiras de mensurar quanto tempo você gastará para fazer uma determinada tarefa, e todas elas são de extrema importância para o funcionamento do mercado. Como já dizia o grande Super Sam, interpretado pelo ator Ramón Valdés (o Seu Madruga): *time is money, oh yeah*.

Porém, uma das coisas que acho mais engraçada no mundo SAP é o tempo desperdiçado com documentações falhas e que nunca serão usadas, servindo somente para fazer volume no "pacotão da homologação". As metodologias dizem que essas tais documentações são importantes, mas a prática nos mostra que, se elas forem feitas do jeito que a maioria dos projetos as crias, elas não são úteis para absolutamente nada. O que diabos acontece com as documentações técnicas em projetos SAP?

As documentações têm muita importância em qualquer projeto de TI. O problema é que o conceito de documentação em vários projetos e empresas que usam o SAP está completamente equivocado. Horas e horas são gastas em documentos que não dizem absolutamente nada sobre os programas criados.

Milhares de palavras gastas para descrever um código-fonte linha a linha, e nem um mísero parágrafo para explicar os motivos que levaram o desenvolvedor a escolher a solução tecnicamente mais complexa para aquele `FORM` crucial do processo. São desenvolvedores brincando de documentar e apro-

vadores felizes com a existência de mais um item para ajudar na homologação do programa.

De volta ao básico

Para que serve uma EF (Especificação Funcional) e uma ET (Especificação Técnica)? Sei que boa parte dos que leem este livro já deve estar careca de saber, mas como o ABAPZombie sempre abrigou muitos iniciantes na área, não custa nada conceituar novamente os dois itens:

- A **Especificação Funcional** (EF) é um documento, criado por funcionais, que explica como um determinado programa terá de se comportar no sistema para atender a uma necessidade de um usuário ou uma empresa. Ele contém principalmente explicações do ponto de vista dos processos de negócio. Pode ou não ter explicações um pouco mais técnicas (nome de tabelas, campos, nome de funções etc.) – isso depende do funcional que as escreve. Em outras palavras, ela deve descrever o que o usuário quer de uma forma que o programador consiga entender (ou, pelo menos, deveria).
- A **Especificação Técnica** (ET) é um documento, criado pelos programadores, que precisa explicar tecnicamente como o software funciona dentro do sistema. É suportado pela EF e contém descrições de como o programador criou o software para atender o que a EF pediu. Normalmente, contém a lista dos objetos criados, uma explicação de como cada objeto foi usado no desenvolvimento, a arquitetura por trás do software e detalhes de como cada parte do programa foi tecnicamente implementada (ou, pelo menos, deveria!).

A criação de ETs ruins se dá em cenários que já se tornaram padrões na prática ABAP. Será que você já se deparou com algum deles?

ETs de 800 páginas para programas de 500 linhas

Muitas empresas têm templates fechados para a criação de especificações técnicas, e a grande maioria delas pedem detalhes demais sobre o programa. Descrições de tabelas campo a campo (colocando o tipo, tamanho, descrição, e outras X bilhões de informações), listagem de todos os `TEXT`s criados no programa com as suas respectivas traduções, atributos técnicos de cada um dos objetos do desenvolvimento e mais outras informações que simplesmente não precisariam estar ali.

Esse é o tipo de ET que tem o menor tempo de vida: é só você mudar a descrição de um campo em qualquer tabela que você criou, e ela já estará desatualizada. É também aquela que não explica quase nada sobre como o programa funciona. Ou seja, se o programador que criou o programa sair da empresa, o outro que se vire debugando até o infinito e além para descobrir como ele funciona.

Criar a ET antes de codificar

Se um mesmo programador vai criar o código e a ET, é muito comum ele primeiro codificar e só depois documentar – mesmo que as metodologias adotadas pelas empresas digam que isso é errado. Porém, existem casos em que um programador mais experiente cria a ET, e um outro menos experiente que codifica com base no documento. Esse tipo de ET tende a ficar com uma qualidade um pouco maior, porque ela explica tecnicamente como o programa deve funcionar sem chegar a um nível de detalhe absurdo e irrelevante para o documento.

O problema é que, na grande maioria das vezes, o ABAPer Júnior acaba fazendo alguma coisa completamente diferente do que estava na ET, simplesmente porque no mundo da programação você só vai saber que certas coisas não vão funcionar como especificadas quando você, de fato, vai lá e tenta codificar.

Esse é um cenário muito comum no dia a dia. O problema é que a nossa classe é preguiçosa e ninguém atualiza a ET após o desenvolvimento ter sido concluído. Ela já nasce desatualizada, e não descreve totalmente como o programa criado funciona.

Fazer da ET uma cópia "sem vergonha" da SE38

No meio da correria do projeto, muitos ABAPers abrem um documento novo no editor de textos, dão `CTRL + C` no código-fonte e um `CTRL + V` na ET, salvam e a entregam junto ao programa. Isso é mais comum do que parece, acredite. E essas ETs passam por todos os níveis de aprovação, acredite de novo.

Criar a DOMM – Documentação Orientada a Mim Mesmo

Muita pessoas esquecem que o monte de coisas descritas em uma ETs servirão como referência no futuro. Em um mundo perfeito, se você chegasse em um projeto para fazer um modificação funcional em um programa de 829.382 linhas, você não teria de debugar tudo para entendê-lo. Primeiramente, iria ler a EF para saber qual processo o desenvolvimento cobre, para depois ler a ET e entender como aquele processo funcional foi implementado pelo programa.

O problema é quando o ABAPer utiliza a ET como um rascunho, um documento onde ele escreve todas aquelas coisas maravilhosas que precisa lembrar sobre o desenvolvimento. O mais legal é que ele costuma deixar tudo descrito da maneira mais desorganizada possível, o que ajuda muito a vida de quem for ler aquilo no futuro (só que não).

Existe solução?

O tempo destinado à criação de documentações precisa passar a ser usado com sabedoria (no maior estilo Jedi mesmo). Não é por acaso que o Manifesto Ágil (http://www.manifestoagil.com.br/) valoriza software em funcionamento mais do que documentação abrangente. O triste é que boa parte das ETs em projetos SAP não serve para nada e só existe porque a metodologia obriga o desenvolvedor a criar o documento para que o produto possa entrar em processo de homologação.

Ninguém questiona o objetivo real de adotar uma metodologia, todos fazem o que ela descreve literalmente, e seguem ou defendem aquilo com uma fé cega, sem avaliar os reais benefícios de todo o processo. No meio corporativo, isso está muito longe de mudar em larga escala.

Particularmente, gosto da ideia de ter um documento que explique o programa, mas odeio a maneira como eles normalmente são gerados. "Ué, então como você acha que podemos gerar documentações que sejam mais úteis?", você me pergunta, jovem (ou velho) Padawan. E eu explico: temos de ser realistas quando tratamos de melhorias em documentações. Não dá para jogar tudo para o alto e dizer que ETs não prestam, que desenvolvimento ágil é legal e que vamos criar nossas próprias documentações do jeito que a gente quiser. Isso simplesmente não vai acontecer em projetos SAP, nem a curto, nem a médio prazo.

Independente de se um dia a maioria dos projetos utilizará metodologias modernas ou não, precisamos melhorar as documentações dentro do cenário atual. Portanto, segue aqui a minha lista de dicas de como criar uma ET que traga valor para o seu desenvolvimento, e que farão o seu tempo ser gasto com alguma coisa que poderá ajudar outros ABAPers no futuro.

1) **Pense no que você gostaria de ler em uma ET alheia.**

   A primeira coisa que costumo fazer ao documentar é pensar nas coisas que um outro programador pode achar interessante se ele ler a minha ET. Quais pontos precisam de uma explicação mais detalhada? Por que eu tive de fazer uma lógica muito mais enrolada em um ponto que, ao se basear na EF, parecia bem mais simples? Por que optamos pelo paralelismo em um programa com performance crítica? Existe algum ponto que é mais fácil de ser entendido olhando diretamente o código? Por que a estrutura de classes foi criada daquela maneira? Se você responder algumas dessas perguntas em sua documentação, ela já valerá muito mais do que a maioria das ETs criadas por aí.

2) **Sem encher linguiça: guarde particularidades do código para o código.**

   Diversas ETs contêm descrições demais sobre partes que só fazem sentido para pontos específicos do código-fonte. Aqui entram criações de variáveis auxiliares, `LOOP`s binários, uso de rotinas de conversão, métodos e funções para conversões de valores, declarações de tabelas internas etc. A não ser que esses pequenos trechos de código façam sentido para o entendimento "geral" da solução implementada, não vale a pena descrevê-los na ET.

3) **Lembre-se: nada de inibir a criatividade do programador.**

   Esse frase foi dita por um grande parceiro meu de projetos e é algo extremamente válido para ETs criadas antes do código. Se você está escrevendo uma ET para um colega menos experiente, cuidado para não cair no mesmo problema anterior: descrever demais o que deve ser criado. Se fosse para você ter que resolver tudo na ET, seria mais fácil ir lá e escrever o código de uma vez, não é mesmo?

   Pessoas mais novas precisam de estímulos para desenvolver sua capacidade de programar, portanto, nada de inibir a criatividade dos seus pupilos. Ah, e peça para ele atualizar o documento posteriormente, deixando a ET fácil de ser entendida por algum outro programador no futuro.

4) **Copiar e colar códigos inteiros na ET: NÃO FAÇA NUNCA!**

   Essa atividade é feita levando em conta a "perda de tempo em criar documentações". Sinceramente, neste caso, até esses 10 minutos que serão gastos dando `CTRL + C` e `CTRL + V` no código serão perdidos. Não faça isso em hipótese nenhuma.

   Quando leio uma ET assim, penso que seu criador não faz a mínima questão de explicar como o seu desenvolvimento funciona para ninguém (talvez porque nem ele mesmo saiba o que está fazendo). Criar documentações faz parte do nosso papel dentro do mundo SAP, e precisamos fazer esse trabalho com responsabilidade.

5) **Faça sugestões de mudança nos templates.**

   Se você não trabalha para um projeto que tenha um template 100% fechado, pode ser de grande valia sugerir mudanças para deixar a ET mais enxuta e de fácil entendimento para outros programadores (ou até mesmo o "você-do-futuro").

   Em projetos que utilizam metodologias ágeis, a documentação costuma ser um pouco mais livre e cada equipe adota somente documentações que eles julguem extremamente necessárias. Se você conhece alguém que trabalha em uma empresa que adota esse tipo de metodologia (mesmo em outras linguagens), pergunte a ele como eles trabalham e o que faz sentido para aquela empresa. Você pode trazer algumas ideias para sua equipe e

fazer a ET, enfim, servir para alguma coisa. É bem raro encontrar empresas que implantam projetos SAP baseados em metodologias ágeis, mas isso não quer dizer que você não possa agregar alguns conceitos em seus trabalhos.

6) **Tente começar a ler ETs de vez em quando.**

Nós, programadores, temos o hábito de sair olhando para o código antes de ler qualquer tipo de documentação. É como quando você compra uma TV nova e nem dá bola para o manual: já sai fuçando e aprendendo tudo sozinho. Para pequenas correções/modificações, faz sentido ir direto no código e alterar; porém, para modificações maiores e restruturação de programas malucos, pode ser que o seu colega programador tenha sido solidário e tenha escrito uma ET bacana, daquelas que vão lhe ajudar a entender como o programa foi implementado. Ler essa ET pode fazer com que você gaste muito menos tempo na sua atividade.

A grande maioria das ETs por aí são descartáveis. Falo isso com certa propriedade, afinal, como você acha que eu consegui mapear esse monte de padrões? Foi lendo ETs pelos projetos afora! Ler essas documentações é legal, porque você vê cada coisa bizarra e também encontra boas surpresas.

É aqui que você que escreve ETs zoadas fica com vergonha ao saber que alguém um dia deve ter lido e xingado seu nome no cabeçalho do documento!

Nem tudo está perdido, depende de você!

Se você tiver que gastar tempo com alguma atividade, procure dar tudo de si. Não enxergue a atividade de documentações como um mero trabalho braçal e inútil, pois ela pode ser muito mais do que isso. Diz a lenda que este que vos escreve descobriu uma certa vontade "obscura" de escrever artigos técnicos a partir das criação de documentações.

Algumas fontes de inspiração para entender o quanto documentações são projetos open source. Experimente analisar projetos populares do GitHub, ou mesmo frameworks populares como o Rails. As documentações costumam ter estrutura intuitivas, são bem completas e fazem de tudo para ajudar os programadores na utilização das ferramentas. De nada adianta criar alguma

plataforma aberta supermega-animal e bonita se ninguém conseguir usar, não é mesmo?

Por fim, falando do mundo SAP, minha recomendação é apertar o `F1` em comandos ABAPs aleatórios e **ler**. A documentação de comandos do SAP é muito detalhada e ajuda bastante se você estiver disposto a dedicar um tempo no entendimento dos termos que, à primeira vista, talvez não façam muito sentido (já que a escrita é bem técnica).

> **Avalie outros pontos de vista**
>
> O tema "documentações" é muito polêmico no mundo SAP. Publiquei uma versão do artigo original do ABAPZombie sobre documentações no SAP Community Network, disponível em http://bit.ly/2bPEyt1. O post gerou uma extensa discussão nos comentários, com outros desenvolvedores expondo os seus pontos de vista (de forma um pouco mais "acalorada" em alguns momentos).
>
> A mensagem do texto é praticamente a mesma, mas vale a pena a leitura de outros pontos de vista nos comentários. Certas pessoas acreditam que um *template* é essencial para o documentações em projetos de larga escala, outras pensam que a automatização por meio de programas geradores de documentação é mais do que o suficiente. Porém, o sentimento principal daqueles que comentaram é o de que as documentações no mundo SAP são mesmo, em sua grande maioria, completamente descartáveis.

6.6 A arte milenar de comentar códigos ABAP

Na seção anterior, discutimos sobre documentações e, em certas partes, sugeri que certas explicações deveriam ficar nos códigos como comentários em vez de inundarem documentos de texto com explicações técnicas detalhadas.

Só que na minhas andanças por códigos alheios e códigos antigos DMM ("de mim mesmo"), sempre acho curioso a quantidade de comentários sem sentido que encontro. Comentar: será que é mesmo **sempre** necessário? Vou contar uma breve fábula sobre comentários, para depois apresentar algumas

ideias para você aprimorar sua técnica nesta arte milenar. Prepare seu kimono e vamos à luta!

Gerando redundâncias inúteis

Quando eu comecei no mundo da programação, disseram-me que "comentar códigos é uma forma de conversar com alguém que dará manutenção no seu programa, ou até mesmo com o seu eu do futuro". Fiquei maravilhado com essa possibilidade de quebrar a barreira do tempo através do ABAP e comecei a comentar freneticamente meus códigos, imaginando que aquele monte de palavras azuis (ou sabe-se lá a cor dos comentários no seu editor) ajudariam alguém algum dia.

O problema é que me falaram que eu deveria "comentar", mas ninguém me disse quais informações exatamente deveria transmitir pelos comentários. E foi assim que começaram a nascer comentários infames, daquele tipo redundante e que não ajuda em nada:

```
START-OF-SELECT

*-- Validações iniciais
PERFORM valida_dados_tela_seleção.

*-- Atualiza dados
PERFORM atualiza_tabela_casdatro.

*-- Exibe ALV na Tela
PERFORM exibe_alv_final.

*-- Análise de todas as ordens de vendas
LOOP AT ordens ASSIGNING <ordem> .

*--Verifica se o Material existe no cadastro de materiais
  READ TABLE materiais WITH KEY
    mantr = <ordem>-matnr
  BINARY SEARCH.

ENDLOOP.
```

Esse tipo de comentário me lembra de provas dissertativas da época do colégio, onde utilizávamos aquela grande técnica de repetir a pergunta que o professor fez, para a resposta parecer maior e aparentar possuir mais conteúdo (quem nunca?).

Com comentários inúteis como esses, acreditava que o meu código estava muito bem comentado e que isso ajudaria muito quem fosse dar manutenção no futuro. Obviamente, comentar o código era um pré-requisito da revisão de pares, logo, quem fosse revisar meu código não poderia reclamar que ele não estava "bem" comentado.

Usei essa técnica falha por mais de 1 ano e meio, até que tive de alterar um código DMM antigo, daqueles que eu havia feito nos primeiros meses da minha carreira como ABAPer. O código era grande, continha diversos problemas estruturais e a lógica era complicada o suficiente para eu ter me esquecido o que diabos aquele pedaço de código fazia. Foi ali que eu percebi que os comentários que eu havia feito não prestavam para absolutamente nada!

Um pequeno exemplo para mostrar o tipo de comentários que encontrei:

```
*-- Análise das Ordens
LOOP AT ordens ASSIGNING <ordem> .

  *-- Busca o cliente
  READ TABLE clientes WITH KEY
    kunnr = <ordem>-kunnr
  BINARY SEARCH.

  IF sy-subrc <> 0.
    *-- Continua para o próximo registro - não soma
    CONTINUE.
  ENDIF.

  soma-vbeln = <ordem>-vbeln.

  *-- Busca o Material
  READ TABLE materiais WITH KEY matnr = <ordem>-matnr.
  BINARY SEARCH.

  *-- Efetua o cálculo
```

```
  ...
  (contas e mais contas)
  ...

  *-- Sumariza os valores
  COLLECT soma INTO calculos.
...

ENDLOOP.
```

Esse código está comentado, mas nem de longe está **bem** comentado. Para que servem exatamente aqueles cálculos? E por que aquela validação de cliente existe? Aliás, como é possível uma ordem ter um cliente que não existe na "tabela de clientes"?

Pessoas mais experientes poderiam encontrar facilmente as respostas para essas perguntas, se fizessem uma análise depurando o código, ou até mesmo lendo a Especificação Técnica / Funcional (contanto que sejam bons documentos). Mas eu era um ABAPer júnior, no meio de um código de 5 mil linhas, com uma ET que não explicava absolutamente nada. E vou lhe falar: sofri muito para entender por que aquela soma estava errada no código. Será que dava para eu ter sofrido menos, se os comentários fossem um pouco diferentes?

Como o "eu" do passado poderia ter ajudado o "eu do futuro"

Todo mundo já fez comentários idiotas no código algum dia. Isso é normal. O que não deve ser normal é você cair na síndrome do robô e continuar fazendo comentários idiotas para sempre.

Muitas vezes somos orientados desde o início a errar, pois aprendemos algumas coisas de maneira errada. Mas a situação que citei me fez pensar em como melhorar os comentários do meu código. Desde então, passei a pesquisar com frequência artigos que analisam a prática de comentar códigos, o que ajudou muito a mudar minha forma de pensar. Há muita informação disponível acerca de comentários, e extraí as dicas mais simples de serem implementadas no ABAP nosso de cada dia.

Começamos nossa jornada de reflexões, jogando no lixo todas as redun-

dâncias:

1) **Se o seu comentário explica exatamente a mesma coisa que alguém entende ao ler o comando, ele não serve para absolutamente NADA.**

   Essa frase já mata praticamente todos os comentários do código do primeiro exemplo deste capítulo. Para que explicar em um comentário que eu vou entrar no bloco que faz validações, se o nome do `FORM` já me indica isso?

2) **Comentários não devem explicar o que o código está fazendo, eles devem mostrar o motivo de aquela lógica existir.**

   Este conceito é uma extensão do primeiro. Usando novamente o exemplo inicial, podemos dizer que aquele comentário "Efetua o cálculo" é um comentário bem inútil. Aliás, em momento algum os comentários explicam por que eu precisei fazer aquelas várias operações: somar valores em função da busca do material, ou ignorar a soma da linha caso o cliente não exista. Eles só dizem o óbvio, aquilo que é possível entender lendo o código puro, são comentários descartáveis.

   Mas então quer dizer que eu posso dar `CTRL + C`, `CTRL + V` do requerimento da EF no código? Diz um velho provérbio chinês que...

3) **...os melhores tipos de comentários são aqueles dos quais você não precisa.**

   Um bom indicador para analisar se o seu código está "limpo" e bem estruturado, é verificar quantos comentários estão espalhados pelo fonte. Normalmente, quando você teve de usar muitos comentários para conseguir explicar o que o seu código está fazendo, pode ser a hora certa para refatorar (ou reescrever) aquela lógica.

A arte de comentar está diretamente ligada a outra arte muito ignorada por programadores: a arte milenar de nomear variáveis e blocos do programa. Com nomes melhores de variáveis e de `FORM`s/métodos/funções, o seu código fica muito mais claro e direto, com relação às suas "maléficas" intenções. No mundo ABAP, todos têm o costume de encurtar o nome das variáveis,

gerando enormes confusões como as famosas `T_AUX1`, `T_AUX2`, `T_AUX3` etc.

Nos `PARAMETERS` e `SELECT-OPTIONS`, há desculpa para os nomes curtos, pois o sistema limita o nome em 8 caracteres. No resto das declarações é possível ser muito mais flexível. Se existe a necessidade de duplicar uma variável, isso quer dizer que cada uma delas serve para uma coisa diferente – e isso pode ser explicado com seus nomes das variáveis.

Para você ter ideia do quanto essa questão de manter o código limpo é importante, há livros inteiros dedicados a essas técnicas, como o livro *Clean Code: A Handbook of Agile Software Craftsmanship*, de Robert C. Martin, Michael C. Feathers e Timothy R. Ottinger.

Nele, os autores defendem a tese de que comentários são necessários somente quando há uma falha do programador em expressar-se através do código-fonte. É uma ideia bem extrema do ponto de vista ABAP, mas perfeitamente viável em diversas situações no mundo da programação.

No nosso caso, quase sempre trabalhamos com desenvolvimentos cheios de regras de negócios complicadas que envolvem tabelas/campos/funções/classes/métodos com nomes em alemão abreviados (se fosse fácil, não teria graça). Basear-se em nomes de campos do dicionário de dados é uma boa prática (como `MATNR`), mas muito deles requerem um pouco mais de explicação (como o elemento de dados `BOSTA` – ele existe, é sério! Veja na transação SE11). Por conta disso, acredito que nós precisamos muito de bons comentários.

De qualquer forma, uma dica legal para você começar a escrever códigos mais autoexplicativos é a seguinte: **programe como se o recurso de comentar códigos simplesmente não existisse**.

Imagine um mundo onde o compilador não reconhecesse o `*` (asterisco) e a `"` (aspa) como caracteres de início de comentários. Um mundo onde você precisa se comunicar com outros programadores e o seu eu do futuro por meio do código. Neste mundo, a tarefa de criar um programa fica muito mais interessante! A não ser que você seja um troglodita, obrigatoriamente você terá que se preocupar fortemente com a estrutura do código, com os nomes de variáveis e de todos os outros objetos.

Entrar neste "mundinho" da próxima vez que você começar a programar

deve ajudar o seu código a ficar muito mais autoexplicativo. A tarefa ficará bem mais difícil, mas a evolução como programador extraída da experiência será permanente. Tento entrar no tal "mundinho" onde comentários não existem e recorro aos comentários para explicar partes complicadas da regra de negócio, ou alguma lógica mirabolante que precisei fazer. O ideal, para mim, é que a pessoa consiga dar manutenções simples no código, sem precisar olhar um documento externo.

A correria dos projetos obriga-nos a criar lógicas monstruosas, cheias de regras malucas e cálculos complicados. Se envolverem regras de impostos e normas do governo, a coisa pode ficar ainda mais confusa. É por isso que gosto muito de explicar de forma geral o que determinado bloco complexo de código faz, como se eu estivesse preparando o "debugador" antes de chegar na lógica pesada.

Costumo fazer isso com duas ou três linhas de comentários, como no exemplo a seguir:

```
* A norma XYZ da Receita obriga que este cálculo seja
* feito com base na lista técnica completa dos materiais.
* Aqui todas as listas são explodidas e as porcentagens
* são contabilizadas no cálculo final.
```

Isso já me ajudou muito quando precisei dar manutenção em programas antigos que eu mesmo havia criado. Agora faça um teste: experimente abrir o código de um programa que você acabou de criar. Quais melhorias podem ser implementadas sem grandes esforços?

A palavra mais importante que você precisa levar em consideração na hora de comentar seus códigos é *didática*. Se os seus comentários forem didáticos ao explicarem os trechos mais complicados do seu código ABAP, você evitará confusões, e ajudará todo e qualquer leitor futuro. Que você possa incorporar essas dicas para desenvolver a didática de comentários na sua prática ABAPer diária, fazendo com que o nível da sua arte atinja níveis absurdos, daqueles em que Bruce Lee nenhum poderia botar defeito.

6.7 Usabilidade em programas ABAP

Existem diversas definições para usabilidade, das mais simples até as mais complexas. Gosto de defini-la como aquela coisa invisível que faz com que o uso do software seja prazeroso, por algum motivo que você não sabe explicar. Aliás, você nem vai pensar sobre o que faz você querer voltar e utilizar aquele software, ele simplesmente **funciona**.

Visualmente, o SAP ERP é um sistema um tanto quanto antiquado. Suas telas estão longe de ser o suprassumo do desenvolvimento de interfaces. Isso é natural, pois trata-se de um sistema com certa idade. A SAP lançou, ao longo dos anos, diversas ferramentas complementares que se conectam ao SAP por meio do ABAP para buscar as informações e mostrar em telas fantásticas, bonitas e fáceis de usar. Entretanto, o seu uso ainda não está completamente difundido (falaremos um pouco mais dessas novas tecnologias e como elas conectam-se ao ABAP no capítulo 8).

Mas, supondo que você é ABAPer e que está desenvolvendo programas utilizando ABAP puro, existe algum necessidade de pensar sobre usabilidade?

Para responder essa pergunta, vamos começar analisando uma situação clássica da vida ABAPer:

1) O ABAPer recebe uma Especificação Funcional para transformar em um programa ABAP *maravilindo*. Muitos dedos são gastos digitando lógicas malucas e cálculos complexos. Depois de ter terminado, ele explica para os funcionais como utilizar o novo programa. O funcional valida a solução e aprova o desenvolvimento.

2) O time funcional explica o programa para o usuário chave, que também valida e aprova o desenvolvimento. Tudo segue até a fase final do projeto.

3) Após a entrada em produção, quando começa o suporte, aparecem erros dizendo que os dados de saída não estão corretos, já que os valores entrados na tela de seleção não estão sendo usados corretamente como filtros.

4) Após um caos momentâneo, o pessoal do projeto *descobre em um passe de mágica* que a maioria desses erros existem porque os usuários **não entenderam completamente** como os programas funcionavam. O time explica para mais usuários (além do usuário chave) como eles devem usar

os programas, e os tais defeitos, para alegria dos gerentes e afins, não são considerados como problemas do projeto.

5) Após alguns dias em produção sem problemas, toda a equipe vai comemorar feliz e contente o sucesso de mais um projeto entregue.

Nesse cenário, todos envolvidos com o projeto vão considerar que o sistema foi implantado com sucesso. Mesmo que o tal usuário que estava usando o sistema de forma errada reclame com seus colegas sobre a má qualidade dos novos programas. Ele passa semanas reclamando que "a sequência de telas não faz o menor sentido" e que "esses filtros parecem filtros, mas não filtros... quem foi o maluco que fez dessa forma?". Depois de um tempo, ele fatalmente para de reclamar, afinal, disseram para ele que o problema era ele, e não o sistema.

O objetivo final de um projeto SAP é prover transações/programas que resolvam um problema do negócio, seja para aumentar a produtividade, ou para tornar possíveis as análises complexas de dados do ambiente. Toda equipe de projeto procura entregar programas que resolvam esses problemas de forma robusta e consistente, com performance adequada e sem erros de cálculos ou regras de negócio.

Agora responda: em qual momento a equipe do projeto procurou imaginar **como** o usuário utilizaria esses programas? Qual a melhor forma de dispor os dados na tela? Quantas telas são realmente necessárias? Esses milhares de botões precisam mesmo estar agrupados nessa tela de seleção? Os termos usados nos textos fazem sentido para o usuário? As mensagens de erro são boas o bastante para que o usuário consiga tomar uma ação corretiva?

Aquele usuário ("o chato", de acordo com o Sr. Projeto), um dos que reclamaram que o programa estava errado, certamente tinha anseios e vontades em relação a usabilidade do sistema que não foram cumpridos pelo projeto. Se você se colocasse no lugar dele, aposto que você também diria que muitas coisas estão erradas, simplesmente porque elas parecem erradas. Usuários normalmente não têm conhecimentos técnicos do sistema, e as suas percepções podem ser completamente diferentes daquelas da equipe técnica.

Às vezes, parece que algumas empresas estão criando software de trás para a frente, pensando primeiro na estrutura do programa e modelagem de da-

dos em vez de pensarem primeiro no usuário. Do que ele realmente precisa? Como ele vai usar a aplicação? Quem tipo de design pode ajudá-lo a ganhar tempo em suas tarefas diárias? Como o design de software pode ajudar a prevenir seus erros? Essas são questões que, infelizmente, (quase) nunca são feitas em projetos SAP.

O importante não é só buscar e mostrar os dados. É importante também pensar em **como** os dados serão apresentados para o usuário. Ele é o motivo pelo qual estamos aqui, e eles devem receber algo que seja fácil e simples de usar.

Um dos grandes erros de vários projetos é ignorar completamente a opinião do usuário quando o assunto é usabilidade. É muito comum ver especificações de programas Online (*module-pools*) com cinco telas, com um monte de informações jogadas, só porque o requerimento funcional diz que aquelas informações precisam estar ali. Como desenhar uma tela sem saber como o usuário final vai usar o sistema em seu dia a dia?

O ABAP não provê muitas maneiras de criar controles de tela; é tudo meio padronizado, exatamente por conta da usabilidade. Mas não adianta nada existir toda uma estrutura visual e de atalhos no sistema, se os programas customizados não seguirem esses padrões de forma que eles façam sentido para os usuários. Um caso simples é colocar o atalho para a ação "voltar" de uma tela em uma tecla de atalho diferente do `F3`. Se você já fez isso, eu **aposto** que a sua orelha já ficou vermelha quando um usuário tentou apertar o `F3` para voltar durante alguma navegação e nada aconteceu. Seu programa pode trazer todos os dados corretamente, mas os usuários vão odiá-lo caso ele não tenha sido estruturado de forma consistente com o resto do sistema.

Os problemas vão além de um usuário aborrecido por um atalho. Pense nas mensagens de erro. Projeto nenhum preocupa-se em mapear e entender os casos onde os erros estarão presentes, achando que um simples "Dados inválidos" é o suficiente para que a aplicação previna um problema.

Esse tipo de mensagem cria um caos ainda maior, já que o usuário precisará descobrir (sabe-se lá como) qual é o "bendito" dado que é inválido. Esse tipo de situação pode gerar um enorme estresse para os usuários do sistema: se ele não entende a mensagem, precisa falar com alguém que saiba, que precisa olhar com ele outros dados, que precisa acessar 76.234 transações. Até eles

perceberem que o problema era um dígito errado na Conta Contábil, quanto tempo levaria para um ABAPer fazer uma consistência melhor na tela de seleção e dizer que somente um dos parâmetros estava errado?

Cansei de ver usuários chave indagando, durante o teste integrado (!), se a mensagem de erro poderia incluir o número do dado inválido para "ajudar a gente". Imagino quantas pessoas deixaram de ser "ajudadas".

Simples ações podem resolver uma boa parte desses problemas. Uma delas é não ter a *usuariofobia*. Tente se aproximar dos usuários, não pense que o funcional é o único que precisa interagir com ele. Você que vai desenhar as telas, criar os controles, implantar os *matchcodes*, agrupar os parâmetros na tela, colocar textos em colunas, traduzir a aplicação, mapear os atalhos, acrescentar *tooltips*, escrever o texto das mensagens de erro.

User eXperience

A facilidade de uso de um software e a simplicidade do seu design são fatores constantemente levantados como determinísticos para que ele seja bem sucedido. O campo da *User eXperience* (UX) analisa como o design de um produto pode afetar as percepções, comportamentos e emoções de um usuário, levando-o a amar ou odiar aquilo que estiver usando. No campo de UX, você conseguirá encontrar os mais diversos estudos sobre como estruturar seu produto (ou software) com o usuário em mente, podendo ser a diferença entre o sucesso e a falha.

Um dos autores mais importantes da área é Donald Norman, escritor do clássico livro *The Design of Everyday Things*. Nesse, ele analisa como o design pode ser usado como uma maneira de comunicação entre objeto e usuário, provendo meios de aplicar técnicas e formas de pensamento para melhorar as suas próprias criações. Recomendo muito sua leitura, pois os conceitos podem ser utilizados não só na criação de programas ABAPs, mas para praticamente qualquer tipo de software.

Na prática, a estrutura de um projeto SAP costuma impedir, por definição, a proximidade do ABAPer e dos usuários – toda a intermediação costuma ser feita somente pelo time funcional. Mas experimente fazer o seguinte teste

(quando nenhum gerente de projeto estiver olhando): pergunte para algum usuário se ele tem algum comentário sobre um programa ABAP que você desenvolveu, só que **não** fale em momento nenhum que foi você quem desenvolveu aquele programa. Um feedback honesto certamente vai surpreendê-lo, pelo bem ou pelo mal (que também é "bem" se você aprender com o erro).

Capítulo 7

Sobre programação sem códigos

Qualquer tipo de programação envolve uma série de considerações que extrapolam a criação do código em si. De nada adianta ter decorado todas as variações dos comandos da linguagem ABAP se você não souber como empregá-las.

Descobrir este "como" requer uma avaliação do contexto do cliente para qual o programa será criado, quais as pessoas envolvidas em todo processo (da concepção ao uso) e das possibilidades de expansão do programa no futuro. Todas as atividades que podem ser executadas antes mesmo de a primeira linha de código ser escrita.

Compartilho neste capítulo uma análise de algumas dessas atividades pré-código, para que você possa exercer a sua função de ABAPer até mesmo antes de efetuar um login no sistema.

7.1 O dilema dos poucos usuários

Certa vez, eu precisava pendurar um quadro na parede da minha casa. Eu, que tinha acabado de sair da casa dos meus pais para morar com minha esposa, não fazia a mínima ideia de como os quadros iam parar na parede. Sim, eu sabia que havia um prego (ou parafuso) por trás, mas nunca havia precisado furar e pendurar um por conta própria (ou vai dizer que você sequer pensava nisso quando morava com seus pais?).

Enfim, peguei a furadeira e furei a parede de casa, em um processo que descascou um pouco de sua tinta, em volta do novo buraco. Não tinha problema, afinal, o quadro iria tampar. Optei por utilizar uma bucha simples para prender o parafuso, mas é claro que a broca que havia usado não tinha o tamanho exato da bucha que eu precisava para prender o parafuso. Portanto, precisei aumentar o furo usando a *mesma* broca (não tinha outra), aumentando consideravelmente a sujeira em minha sala.

Depois de uma série de tentativas, finalmente encaixei a bucha e consegui fixar o parafuso. Pendurei o quadro – que não era muito pesado – e nunca mais precisei alterá-lo. Ele continua ali na minha parede, e nunca mais me deu trabalho.

Supondo que, no futuro, eu queira colocar um quadro muito mais pesado na parede, essa gambiarra que eu fiz vai funcionar? Há alguma chance de eu ter aberto o furo demais e o quadro despencar? Precisarei fazer tudo de novo, aumentando o furo ainda mais para colocar uma bucha e parafusos mais adequados para um peso maior?

Ah, nada disso importa. Eu ainda gosto muito do quadro e não pretendo tirá-lo de lá tão cedo.

Nesse pequeno exemplo – que demonstra que não nasci de forma alguma para trabalhos manuais –, preferi não me estressar tanto ao avaliar qual a forma perfeita de criar um mecanismo resistente para segurar qualquer tipo de quadro que eu venha a colocar no futuro. Tudo o que eu queria era pendurar o quadro e pronto.

O **tempo** que eu gastaria gerando a solução perfeita para qualquer quadro do mundo não se justificava, pois o **aumento do benefício** que eu teria por ter feito a mega solução de quadros seria bem pequena em relação ao meu "furo aumentado na marra".

E se eu decidisse que não quero trocar de quadro, não vou precisar me preocupar mais com essa parede pelo resto da vida?

Diversas vezes, vi projetos entregando programas ABAP com a mesma mentalidade que eu tive ao pendurar meu quadro. Afinal, pouquíssimos programas ABAP são usados em larga escala nas empresas. Ou seja, não é preciso se preocupar com telas de seleções robustas, mensagens claras e objetivas, e uma arquitetura de código que permita extensões pouco traumáticas.

O **tempo** que levaria para gerar um programa bom não se justificava, já que o **aumento do benefício** que seria adquirido não poderia ser resgatado diretamente naquele projeto. Entregar algo que funciona, para algum tipo de uso pontual, torna-se o suficiente.

Acontece que programas em empresas não são iguais aos quadros que temos nas paredes das nossas casas. As regras de negócio podem mudar a cada nova semana, áreas corporativas podem crescer da noite para o dia, e aquele programa que era usado por alguém em uma sala escura pode, subitamente, transformar-se na base para uma série de atividades que envolvem muitas pessoas.

O dilema de poucos usuários é constantemente usado no mundo SAP como desculpa para realizar um mau trabalho. O imediatismo da entrega cria um sistema no qual um trabalho mediano é mais do que o suficiente. Infelizmente, os impactos podem ser catastróficos.

Qualquer ABAPer com pouco tempo de experiência já precisou debugar algum programa monstro de milhares de linhas (Z ou um standard) que faz parte de um processo crucial para a empresa. Será que, no processo de criação daquele código, ele foi feito para ser usado para uma grande parcela de usuários, ou somente para uma minoria bem específica? Dada a falta de documentação, cada novo programador precisava criar o famoso "puxadinho" no código, criando uma infinita colcha de retalhos lógicos ininteligível?

Por mais bizarro que seja o requerimento, implantá-lo em um programa monstruoso é sempre mais difícil. Ao contrário da sabedoria popular, gerar uma solução melhor nem sempre levará mais tempo.

O caminho mais fácil costuma ser o caminho braçal, ou seja, você resolve muitos problemas escrevendo muito código, sem ponderar qual a melhor estratégia de implementação. **O caminho mais inteligente** costuma precisar

de menos código, você passará mais tempo pensando em vez de escrevendo ABAP.

Bom, aposto que você não quer correr o risco de plantar uma semente de monstrengo no seu código, direcionando-o somente a poucos usuários. Para reduzir as chances, é preciso pensar em algumas coisas antes mesmo de abrir o editor ABAP e escrever os primeiros comandos.

7.2 Duvide do caminho mais "fácil"

Quando uma nova especificação funcional aparece em seu e-mail, o que você faz? Por trás desses documentos existem prazos, expectativas, estimativas, dinheiro e pressão. É natural que a primeira reação seja abri-la e iniciar imediatamente a criação do programa, para poder se livrar rapidamente do gerente que não para de perguntar "qual é o status?". As chances de um começo precoce são ainda maiores se existir algum "direcionamento" técnico na Especificação Funcional, definido sabe-se lá por quem.

Algumas páginas atrás, no capítulo 5, refletimos se todo ABAPer é realmente um consultor. O problema aqui é bem parecido: a estrutura clássica dos projetos SAP contribui para que você não seja envolvido no processo de arquitetura do sistema, portanto, seu único input sobre o programa a ser criado é enviado pelo funcional através da EF.

Essa EF muitas vezes contém as famosas "sugestões de implementação técnica", uma pseudoarquitetura do código que não foi criada por nenhum programador. Quem cria a EF é um funcional, alguém que "deve" possuir bons conhecimentos das áreas de negócio da empresa e sabe como traduzi-los para processos dentro do SAP ERP. Processos, não **códigos**.

A EF pode conter inúmeros detalhes do negócio a ser automatizado, mas o funcional não tem a obrigação de possuir grandes conhecimentos sobre como um determinado programa deve ser criado. Ele não sabe qual a melhor estratégia de implantação, qual o melhor tipo de objeto a ser criado, qual a melhor tecnologia para resolver o problema, ou qual técnica de programação será empregada.

Decidir isso tudo é responsabilidade do ABAPer. Claro que, na prática, alguns funcionais podem até facilitar o trabalho técnico por terem algum tipo

de conhecimento relacionado (seja por conta da experiência, ou por cargos anteriores), mas, mesmo assim, qualquer diretriz técnica como EF serve somente como uma sugestão, e não como verdade absoluta.

O caminho fácil de usar uma sugestão dessas como base pode ter resultados catastróficos. O maior problema desse cenário revela-se quando o ABAPer decide que tomar esse caminho lhe dará um passe livre para abster-se de problemas futuros. "O problema X no método Y só acontece, pois o código está igual ao pedido na EF": esse tipo de empurra-empurra de responsabilidades nunca vai resultar em algo benéfico para o projeto, o programa e as pessoas envolvidas, além de revelar um total descaso do desenvolvedor com o programa sendo criado.

Se ter envolvimento com as fases de definição de um projeto não é tarefa fácil para ABAPers alocados ou trabalhando diretamente para clientes, imagine então para os que trabalham em uma área muito comum nas consultorias: a fábrica de software.

A ideia de uma fábrica é otimizar a mão de obra do desenvolvimento, fazendo com que os ABAPers que ali trabalham possam atender demandas de mais de um cliente, sem ficarem atrelados à duração completa de um determinado projeto. Essa otimização dificulta ainda mais o acesso ao contexto do projeto, e o desenvolver acaba criando programas para resolver problemas que ele não sabe quais são.

Já recebi muitas dessas "sugestões técnicas" em EFs quando trabalhei em fábricas, e confesso que era tentador apoiar-me completamente na solução desenhada pelo funcional na EF, quando eu estava iniciando com ABAP e programação, em vez de refletir junto a alguém mais sênior sobre uma melhor estratégia de implementação. Sem uma visão completa do projeto, sem a experiência para prever erros futuros e com pressão de entrega por todos os lados, é fácil cair nessa armadilha.

Uma outra armadilha muito comum é sair criando cópias e mais cópias de programas Z, standards ou de internet, sem o menor critério de avaliação. Seja pela EF ou através do cliente, aparecem ordens como "faça um cópia do programa `Z_MOSTRO_BIZARRO` e altere *só* um `FORM`".

Verificar se algo faz sentido antes de copiar é algo básico; entretanto, não foram poucas as vezes em que encontrei conjuntos de 4 ou 5 programas di-

ferentes que não passam de cópias um dos outros, com algumas pequenas modificações. Em tempo de desenvolvimento, isso pode parecer benéfico, mas no momento em que alguma funcionalidade comum parar de funcionar, o ABAPer terá que alterar todos os 5 programas.

Todo programador é (ou deveria ser) ensinado a criar soluções reutilizáveis (das quais falaremos a seguir). Porém, é impressionante como as boas práticas são jogadas no lixo em ambientes corporativos, em prol dos prazos de entrega dos projetos.

Independente do nível técnico de uma EF, a responsabilidade da implementação será sempre do desenvolvedor. Pular a fase essencial, que é planejar e arquitetar a implementação do software, certamente resultará em falhas.

Uma boa alternativa é questionar o funcional, entendendo com ele os motivos que o levaram a adicionar tantos detalhes técnicos na EF – a intenção normalmente é de ajudar em vez de atrapalhar. Se a intenção foi positiva, um *brainstorm* conjunto torna-se uma ótima alternativa para maximizar a qualidade da arquitetura.

Desconsiderou o caminho fácil e chegou em uma solução interessante, levando em consideração o processo descrito na EF? Então vamos continuar as análises, pois ainda não estamos prontos para escrever nenhuma linha de código.

7.3 Mesclando o velho com o novo

Já falamos bastante sobre a retrocompatibilidade e seus impactos no capítulo 6. É possível criar um desenvolvimento ABAP hoje ao usarmos features bem velhas até novidades implantadas nos últimos Support Packages, desde que o sistema esteja devidamente atualizado. Então, o que utilizar: o velho ABAPão de sempre, ou o ABAP-hipster dos tempos modernos?

Vamos supor que você programe (ou tenha aprendido a programar) do mesmo jeito que se programava ABAP nos anos 90, e não pretende mudar sua forma de trabalho. Seus programas estão regados de comandos marcados como obsoletos na documentação oficial, a estruturação interna é completamente baseada em `FORM`s e `CALL FUNCTION`s, e classes são utilizadas somente quando é feito um `ALV` (usando um *copy-paste* da internet, é claro).

Apesar do trabalho à moda antiga, sua experiência compensa a falta de atualização técnica, já que você conhece milhares de manhas para driblar o sistema e conseguir fazer aquilo que precisa, dentro de um tempo controlado.

Após entregue, manutenções pontuais tendem a ser simples, afinal, você programou mais do mesmo. Já as manutenções grandes podem ser bem complicadas se a estruturação do paradigma procedural não foi bem aplicada (o que geralmente acontece). Entretanto, do ponto de vista de comandos, todos terão base para entender o que foi criado. A reutilização de componentes geralmente é nula, o programa funciona para aquele requerimento específico e pronto.

Agora, imagine que você absorveu todo o turbilhão de informações novas sobre as últimas versões do ABAP e está pronto para aplicar absolutamente tudo. Obviamente, seus programas são criados utilizando ABAP Objects, gerando classes a partir de diagramas elaborados que contemplam claramente os processos requeridos na EF.

Diversas novas features da linguagem são implantadas nos seus códigos – daquelas que nem 10% dos seus amigos ABAPers conhecem –, eles procuram seguir diretrizes modernas de programação e abominam o *jeito procedural de ser*. Na hora de ajustá-los, a primeira reação dos outros desenvolvedores será um misto de espanto e medo. Espanto porque seu código pode parecer ter sido escrito em élfico para alguém que não acompanha com tanto afinco atualizações da linguagem e maneiras extra-SAP de trabalhar; e medo por não saber se uma mísera alteração de um IF naquele programa megaelaborado pode destruir completamente o sistema.

Apesar da possibilidade de xingamentos por parte de alguns pares, o efeito colateral de ter se preocupado em criar um código robusto é que a reutilização da solução para outros problemas será bem grande. Isso possivelmente gerará grandes otimizações de tempo em futuros projetos.

Usar tudo que há de bom e melhor no mundo da programação e do ABAP resultará em um programa superior à abordagem de programar "mais do mesmo". Acontece que, se a maioria dos ABAPers que forem dar manutenção nesse código não entenderem absolutamente nada sobre os novos comandos, ou estiverem desatualizados quanto às práticas de programação, essa abordagem pode gerar um monte de transtornos, pois tudo vai levar muito mais

tempo do que o previsto para ser ajustado/corrigido.

"Aquele que vai alterar que corra atrás e aprenda!". Uma abordagem como essa é benéfica quando todos que farão manutenção nos códigos estiverem dispostos a aprender e tiverem tempo para tal. Infelizmente, uma série de fatores pode fazer com que determinada equipe de uma empresa decida erradicar completamente tudo o que é novo do seu sistema, do ponto de vista de programação. Explico com uma situação que já enfrentei em mais de uma empresa.

Nivelando um sistema pelo passado

Certa vez, um entusiasta da modernização do ABAP decidiu criar uma série de programas gerados em um projeto utilizando todas as atualizações da linguagem. Lembre-se aqui de que ABAP está na versão 7.4 (quando da escrita deste livro), permitindo o uso de comandos bem antigos com a retrocompatibilidade. Não é exagero algum dizer que um ABAP "das antigas" não conseguiria entender praticamente nada dos programas que o entusiasta criou. E foi exatamente o que aconteceu, de uma maneira um pouco trágica.

O entusiasta criou seus programas modernos durante sua alocação em um projeto, que durou até o suporte pós-implantação em produção. Acontece que, alguns meses após esse projeto, uma alteração em outros programas que subiu para produção gerou um erro em cadeia (afinal, no SAP praticamente tudo é conectado de alguma forma), exigindo que os programas criados pelo entusiasta fossem alterados para evitar uma falha crítica em produção.

A tarefa de ajustar os programas precisava ser feita com extrema rapidez, pois o problema impactava diretamente a entrada de matéria-prima na empresa. Se não fosse corrigido em questão de horas, havia um risco de a linha de produção parar, resultando em perdas significativas de dinheiro.

A alteração não era nada de outro mundo e o programa estava estruturado para suportar um ajuste desse tipo. Tendo sido criado usando ABAP Objects, uma alteração nas condições para gerar uma instância de uma determinada classe que implantava o *design pattern* `Factory` (uma construção clássica da Orientação a Objetos) seria o suficiente. Porém, o ABAPer do cliente não fazia a menor ideia do que era OO, quanto mais um design pattern.

O resultado não poderia ser outro: o ABAPer culpou a "modernidade do

código" pela demora da alteração. Após milhares de ligações de gerentes e diretores descabelados, encontraram alguém que entendia de ABAP Objects para ajustar o programa. Entretanto, isso não foi o bastante para apagar a percepção de que o código "ABAP de sempre" era o código que realmente era bom, e que ter aplicado coisas "novas demais" tinha sido a causa raiz de toda a confusão.

O resultado? O ABAPer do cliente gerou uma restrição para todos os futuros ABAPers que prestassem serviço naquela empresa: "Não inventem nada, façam todos os códigos do mesmo jeito que sempre foram feitos desde 1990". Essa diretriz perdura até hoje.

O ABAPer do cliente, por ser um desenvolvedor de código, deveria ter empenhado-se para entender o que estava escrito ali, para então fazer a alteração necessária. É um tremendo absurdo que ele tenha jogado a culpa na modernidade do código, não é mesmo? Fazer uma afirmação a essa pergunta de forma pragmática é bem fácil, mas a coisa toda complica se considerarmos a pressão depositada em cima daquele ABAPer naquele cenário.

Alguns ambientes corporativos podem se tornar extremamente nocivos em situações de caos, como essa. Eu realmente acredito que a culpa foi do ABAPer que não sabia compreender o código, mas isso não quer dizer que eu não entenda os motivos que o levaram a tomar uma decisão errada, jogando o erro para a prática moderna.

Se toda essa história parece maluca demais, peço para que você faça um teste e pergunte aos seus amigos e conhecidos ABAPers quantos deles sabem quais as grandes adições feitas à linguagem ABAP em suas últimas versões. Ou mesmo quantos tiveram uma formação ABAP que ensinava algo além da programação procedural, ou citava a existência do mundo além das ferramentas básicas do ABAP, como o ALV (S**A**P **L**ist **V**iewer – o motivo de terem usado o A em vez do S para a primeira letra da sigla é um eterno mistério).

O grande dilema é que, mesmo que alguém tenha contato com o ABAP moderno durante sua formação, a realidade dos projetos e clientes com seus sistemas desatualizados e sem ferramentas devidamente instaladas força que o nível do ABAP seja sempre nivelado pelo passado. Situações extremas, como clientes que proíbem o uso de práticas modernas, são só mais um dos sintomas, e não a causa raiz.

O standard não foi construído em um dia

Felizmente, muita coisa tem mudado no mundo SAP nos últimos anos. A empresa vem fazendo campanhas enormes para que seus novos produtos (dos quais falaremos no capítulo 8) sejam comprados, implementados e usados por toda sua base instalada. No longo prazo, novas features do ABAP serão obrigatórias para que um determinado projeto seja sequer implantado.

Mas como fazer para inovar hoje, aplicando o que há de novo sem medo de criar um caos corporativo generalizado?

Minha sugestão é criar uma escala de inovação, no momento de decisão sobre qual caminho tomar ao criar o design e arquitetura do(s) seu(s) programa(s):

- **Inovação total e sem frescuras:** se a equipe do cliente (ou empresa na qual você trabalha) mantém uma cultura interna de atualização constante dos seus sistemas internos, use e abuse de todas as novidades que o sistema permitir, sem preocupações.

- **Índices da tendência de fazer mais do mesmo:** se a equipe não contém nenhuma restrição, mas contém membros que demonstram certo receio quanto ao uso de novas ferramentas e comandos da linguagem, ainda assim implante o que você julgar necessário. Porém, deixe tudo muito bem documento e explique pessoalmente a todos que forem dar manutenção sobre o que você fez. Dessa forma, você fomenta discussões sobre atualizações do ABAP, futuro da linguagem e desafios na adequação de coisas novas em sistemas antigos.

- **Restrições retroabsurdas:** mesmo em cenários onde exista uma situação de restrição com inovações, ainda assim aplique algum conceito novo – mas vá com calma. Tentar provar que o uso de uma nova biblioteca de exibição de telas como o *SAPUI5* através do *Gateway* (falaremos de ambos no capítulo 8) é falha quase certa. Alguma pequena modificação no código – como declaração de variáveis *inline* – prova-se eficiente, e é fácil de ser entendida por qualquer ABAPer. A inserção de pequenas atualizações é uma das únicas formas de reverter uma situação tão proibitiva, e é uma atividade que precisa ser feita de forma constante para surtir algum efeito.

Aliada a essa lista, crie uma meta pessoal de tentar implantar uma pequena nova feature a cada um dos programas que você fizer. Aposto que após algumas pequenas atualizações se provarem úteis, você passará para algumas maiores.

Por mais cômodo que seja assumir que todos deveriam estar atualizados e deveriam utilizar o que há de mais moderno na linguagem, não dá para ignorar o fator humano de movimentações como essa em uma comunidade que, em parte, não vê motivo algum para deixar de trabalhar do jeito como sempre trabalhou.

Apoiar-se no comodismo que a retrocompatibilidade traz faz com que ABAPers se afundem cada vez mais no abismo da incerteza se as suas habilidades continuarão válidas diante todo o novo ecossistema de produtos que a SAP vem criando nos últimos anos. A melhor forma de combater esse tipo de incerteza é inovar de forma contínua, com um passo de cada vez e de acordo com as diretrizes do cliente onde o trabalho será feito.

7.4 Reutilização local ao em vez de cópia externa

Quando falamos de soluções que podem ser reutilizadas no mundo SAP, logo se pensa em programas que podem ser portados de um cliente para o outro, como "aceleradores" do desenvolvimento. O cliente X pede um relatório de vendas genérico, que possivelmente poderá ser utilizado no cliente Y (desde que o projeto seja vendido). O ABAP, então, copia o código gerado no cliente X, a fim de usá-lo no futuro como um acelerador, caso o cliente Y, W ou Z venham a ter tal necessidade no futuro.

Essa é uma prática corriqueira, ainda mais quando pensamos que o SAP ERP consiste de instalações separadas por clientes que, muitas vezes, pertencem a um mesmo segmento e certamente precisarão suprir deficiências semelhantes ao processo standards, com códigos ABAP customizados. Eventualmente, alguns desses aceleradores são melhorados com o tempo e acabam virando produtos. Há até empresas que sobrevivem primariamente da comercialização de produtos criados usando o ABAP.

Entretanto, criar um programa em um cliente e copiá-lo para vender em

outro é meramente uma visão comercial do processo. Ela não é *errada*, mas tende a atropelar a busca pela qualidade de código em prol da qualidade do dinheiro. Vamos ignorar esse pensamento por hora, e focar na reutilização de código em uma mesma instalação/cliente.

Recapitulando o que já exploramos anteriormente, uma instalação do SAP ERP contém um número imenso de programas ABAP entregues ao cliente pelo time da própria SAP. Junto a esses programas standard está uma série de funções, classes e outros componentes ABAP que podem ser incorporados em programas Z, gerados pelos ABAPer que customizam o sistema. É comum que ABAPers apoiem-se *somente* nos componentes standards gerados pela própria SAP para realizar tarefas comuns, como: download/upload de arquivos, geração de componentes de tela ou manipulação de processos funcionais do sistema.

O exemplo mais clássico é a ferramenta que gera os famosos relatórios ALV. Há três formas de usá-la em programas ABAP: através de funções, como a `REUSE_ALV_GRID_DISPLAY`; através da classe mais popularmente utilizada, a `CL_GUI_ALV_GRID`; ou através da classe mais atual, a `CL_SALV_TABLE`.

Se um ABAPer precisar criar um ALV em um cliente, normalmente ele irá diretamente a uma dessas classes, mesmo que algum outro ABAPer tenha criado algo mágico e mais viável para aquele caso, naquela mesma instalação. Explico.

Imagine que um ABAPer foi até um cliente e precisou fazer um projeto para implantar cinco relatórios ALV, cada um com uma visão diferente de informações das vendas dos produtos. Ele sabia que os relatórios seriam praticamente iguais, mudando somente alguns cálculos complexos e a formatação dos dados para exibição e filtros. Em vez de criar 5 vezes a mesma lógica que implantaria a mesma seleção e a lógica de criação do ALV, preferiu criar uma classe para tratar tudo que era comum, abstraindo a lógica funcional para ser embutida separadamente em cada um dos relatórios.

Ele criou métodos para receber os filtros, usando técnicas de associação dinâmica (como o `ASSIGN` e as classes do *Runtime Type Services*) para resolver os tipos de cada variável, e métodos para receber parâmetros de exibição. Também implantou uma hierarquia de classes, de forma que era possível de-

rivar a classe principal que geraria o ALV para uma "classe específica do relatório X". Assim, os cálculos poderiam ser implantados reaproveitando todo o mecanismo principal.

Os usuários precisavam de 5 programas diferentes, por conta de permissões de acesso e telas de filtro diferentes. Logo, o ABAPer finalizou criando 5 programas, todos utilizando sua classe geradora de ALVs para relatórios de vendas.

Se meses após a criação desses 5 relatórios surgissem mais 2 que, por algum motivo, precisassem ser feitos por outra consultoria, com outro ABAPer, será que ele usaria a mesma classe para otimizar o processo, ou recorreria às implementações standards da SAP?

É muito difícil controlar uma base de códigos no detalhe, em um ERP de uma empresa gigantesca. Não estamos falando de 100 ou 200 códigos, mas de centenas de milhares de códigos-fonte em alguns casos. Por ser tanta coisa, é impossível que o ABAPer que fosse implantar os 2 novos relatórios sequer soubesse da existência dos cinco primeiros.

O cliente poderia até contar para ele dos 5 relatórios anteriores, porém ele quase nunca sabe detalhadamente qual foi a abordagem técnica que o time de desenvolvimento adotou na criação dos programas. Isso porque as tarefas são delegadas para as consultorias ou equipes internas, na esperança de que a documentação seja o porto seguro para entender o que foi feito. Conforme vimos no capítulo 6, a maior parte das documentações ABAP de grandes empresas não explicam absolutamente nada sobre os códigos.

O ABAPer da nova consultoria provavelmente recriaria tudo do zero para os 2 novos relatórios. Com um prazo nas costas geralmente apertado, ninguém vai ficar fuçando milhares de fontes para conseguir encontrar uma classe que foi feita pensando na "reutilização de código para relatórios do processo de vendas".

Percebem o quão específico isso soa? Seria como procurar uma agulha em um palheiro gigante, com um prazo definido em horas. Você pode dar a sorte de achar em poucos minutos, ou gastar todo o tempo disponível procurando e acabar sem nada. Aliás, sem nada não, provavelmente com alguém crucificando sua lerdeza em entregar o programa pronto.

Quem sai mais prejudicado de todo esse rolo é o cliente que, por não con-

seguir controlar exatamente como os programas são criados, acaba com várias versões diferentes para a mesma solução implantadas em seus programas ABAP.

Já vi desenvolvedores de outras tecnologias, acostumados com o gerenciamento de aplicações específicas, assustarem-se com esse volume de programas ABAP criados, que aparenta ser um caos que não faz o menor sentido. Esse sentido existe, mas está extremamente fragmentado com áreas distintas da empresa, responsáveis por pequenas partes do processo, todos embolados nesse repositório imenso do qual o ABAP pode ter visão do todo. Qualquer um se assustaria.

Com esse exemplo, fica claro o motivo de o ABAPer sempre recorrer ao standard quando falamos de reutilização. Mas, se por um lado isso dá mais segurança do que o código standard reutilizado – que em teoria é mais robusto –, por outro cria um modelo mental de não criar pensando na reutilização local dos componentes do seu programa.

Tudo é sempre refeito do zero e, talvez, copiado para um outro cliente. É esse tipo de abordagem que faz com que a maioria dos programas criados em instalações do ERP seja feita pensando em suportar somente um quadro pela vida toda na parede.

Um dos conceitos mais básicos do mundo do desenvolvimento de software é o DRY – *Don't Repeat Yourself* –, cunhado por Andy Hunt e Dave Thomas, no livro *The Pragmatic Programmer*. Todo programador aprende desde cedo que é preciso organizar o seu código de forma otimizada, pensando em uma estrutura que potencialize o reúso e eliminando a necessidade de escrever um mesmo bloco de código repetidas vezes.

Mas por que esse conceito é aplicado somente na escala "local" de cada um dos programas ABAP, e não na instalação do ambiente como um todo?

Talvez esse seja um dos principais motivos da enorme resistência que o paradigma da Orientação a Objetos enfrenta no mundo ABAP: pensar que cada pequeno programa trabalha de forma separada, sem ser necessário criar pensando no reúso por parte do sistema como um todo. Algum tipo de reutilização de código precisa existir somente para aquele pequeno programa feito no projeto, que resolve um pequeno problema e ponto.

O mais irônico é que, se você cria um programa ABAP pensando no reúso

– do ponto de vista de todo o sistema –, sua solução será **automaticamente** portável para qualquer outra instalação SAP, desde que isso seja feito de forma legal (veja o box a seguir para mais informações).

Naquela história dos cinco relatórios, eles poderiam facilmente virar 50. Talvez não no mesmo cliente, mas pense nas possibilidades se a lógica principal dos relatórios fosse portada para outros lugares. Com alguns ajustes e melhorias, uma pequena ideia como essa de relatórios poderia virar algo maior e, quem sabe, até mesmo criar um caminho para uma empresa de produtos.

Mas mesmo que você não queira dominar o mundo com programas ABAP, tente pensar na utilidade de partes do seu programa do ponto de vista de todo o sistema. Quais partes poderiam ser reutilizadas em mais de um programa? Qual a melhor forma de criar componentes técnicos que abstraem a lógica funcional? Como documentar/nomear os componentes de forma que eles sejam mais bem entendidos por outros ABAPers (e até por você mesmo no futuro)?

Você não precisa iniciar uma abordagem deste tipo, tentando abstrair um processo funcional descrito em 374 páginas. Comece identificando padrões em pequenas lógicas e crie mecanismos para que você não precise nunca mais fazer aquilo novamente.

Qual a legalidade da cópia de códigos entre clientes?

Na prática, sempre existiu a cópia deliberada de programas ABAP de um cliente para o outro no mercado e, normalmente, ninguém se preocupa com os detalhes que indicam se isso é uma prática legal ou não.

Se você é contratado por um cliente para criar um determinado código, isso indica que o código gerado é de propriedade do cliente, e não sua ou da empresa para a qual você trabalha. Entretanto, há algumas formas de tornar esse tipo de compartilhamento legal: através dos *namespaces* (espécie de "pacotes" criados pela SAP para parceiros e clientes, que dá a "benção" para que os códigos sejam portados pontualmente/como produtos); ou baseando-se nos termos descritos no documento *SAP Developer License Agreement* (http://bit.ly/2byZw17) . Este último, felizmente, permite que códigos e ferramentas ABAP feitos em ambientes Trial ABAP sejam livremente compartilhados por seus autores, por meio de licenças open source.

Ferramentas clássicas do ABAP open source

Algumas pessoas extrapolam o pensamento do reúso em projetos e elevam suas ideias ao nível de reúso por toda a comunidade. Há duas ferramentas ABAP *open source* que considero essenciais para o dia a dia: o **SAPLink** e o **ABAP2XLSX**, ambos criados e mantidos por uma força tarefa que envolve ABAPers dos mais diversos cantos do mundo.

O **SAPLink** (http://www.saplink.org) facilita a movimentação de objetos ABAP (telas, funções, classes, programas, onlines e tudo mais) entre dois sistemas diferentes, sem a necessidade de envolver a equipe de BASIS e o transporte de requests.

Já o **ABAP2XLSX** (http://www.abap2xlsx.org/) permite a criação de planilhas do Excel extremamente complexas a partir de um conjunto de classes, simples e intuitivo. Além de serem ferramentas muito úteis, servem de inspiração para que você possa elevar a aplicação do reúso em seus próprios projetos.

7.5 A caverna de fones

De tudo o que discutimos neste capítulo, a parte que considero mais difícil não é aprender a melhor forma de reutilizar soluções, testar novas tecnologias ou fazer experimentos com projetos open source. Para mim, a parte mais difícil é, de longe, sair da caverna de fones.

Se você já teve a oportunidade de trabalhar em projeto em um cliente, sabe que alguns desses ambientes não são lá muitos próprios para a arte de programar. Há muito barulho, muita conversa e muitas distrações.

O último reduto dos ABAPers acaba sendo os fones de ouvido, que nos levam para dentro do nosso "mundinho". Lá, uma música, rádio ou podcast ajudam a simular uma desconexão com a realidade, aumentando nossa capacidade de focar em montar a lógica de nossas criações. Somos, enfim, capazes de focar nossas mentes na programação.

Entretanto, note que existe uma sinergia imensa entre todos os tópicos discutidos anteriormente: você sempre precisará falar com alguém, seja o funcional, o usuário, o cliente, a comunidade ou o seu amigo ABAPer.

Parece maluquice, já que todos nós claramente nos comunicamos. Entretanto, lembre-se de que discutimos lá atrás, no capítulo 5, que a desconexão entre o ABAPer e o usuários imposta pelos projetos impede que os ABAPers realmente prestem consultoria nos projetos por onde passam, ficando reféns da interface com os funcionais. O quanto disso é culpa estritamente da estrutura imposta pelos projetos? E o quanto é culpa dos próprios ABAPers que ficam fechados em seus mundinhos, esperando que alguém mande que ele faça alguma coisa?

O problema não é o fone em si, mas é a ideia de que é possível ser um programador ABAP sem se envolver ativamente em discussões e definições. Se o SAP ERP, por definição, é um sistema completamente integrado, pensar que aqueles que trabalham neste mundo podem sempre trabalhar de forma fragmentada é um erro.

Tudo o que você faz para melhorar como programador precisa ser efetivamente testado. Quanto mais rápido você puder conversar com alguém para testar se as suas abordagens estão certas ou não, mais rápido você vai melhorar.

Em vez de somente crucificar um funcional por colocar código na EF, converse com ele sobre isso o mais rápido possível. Discuta sobre inovações diretamente com o cliente, gerente, líder ou seja lá quem puder ajudá-lo a implantar alguma ideia nova que você tiver. Consulte outros desenvolvedores para saber se a sua abordagem técnica favorece algum tipo de reúso, ou se você esqueceu alguma coisa que poderia ser melhorada.

Seja por poucos minutos, ou por algumas horas, saia da sua caverna de fones e explore essas abordagens. Nem sempre você conseguirá implantar tudo de uma só vez, mas não tem problema. O importante é dar o primeiro passo, e não cair em um marasmo eterno que o deixará apagado e com poucas possibilidades de evoluir aplicando todas as suas capacidades como ABAPer. Considere tudo isso para depois, finalmente, abrir o editor e programar.

Capítulo 8

O que vem por aí?

Quando comecei a trabalhar com ABAP, era comum escutar nos corredores de consultorias que o ABAP morreria e daria lugar ao Java. Até hoje devem existir pessoas desinformadas que acreditam em tais rumores. Pode ser que o ABAP pareça defasado, afinal, existem tantos produtos e tecnologias novas que o ABAP fica um pouco distante dos holofotes.

Apesar de todos os novos portfólios de produtos, é fato de que o SAP ERP ainda é o carro-chefe da empresa alemã. O foco de vendas está voltado para novidades, mas tudo relacionado ao ABAP é o que manteve e (também) mantém a empresa funcionando a pleno vapor.

Portanto, não é surpresa nenhuma descobrir que, por baixo de milhares de siglas e conceitos novos, o ABAP quase sempre está presente de alguma forma. Por esse motivo, diversas evoluções estão sendo implantadas na linguagem, muitas delas visando a simplificação da escrita, integração nativa

com produtos novos da SAP e suporte a técnicas disponíveis em linguagens de programação modernas.

Para demonstrar um pouco das oportunidades relacionadas ao ABAP no futuro, destaquei o envolvimento da linguagem com três novos produtos SAP que têm grandes chances de aparecerem no seu dia a dia nos próximos anos. Todos servem de base para outras soluções SAP complementares, que inclusive já possuem uma base considerável de casos de implantação pelo mundo.

Isso pode ser visto pelo site oficial do HANA (http://hana.sap.com/) , carro-chefe dessa nova geração, no qual é possível acessar uma lista com diversos casos de uso em empresas "gigantes" dos mais diversos segmentos: Ebay, Mercedes, Vodafone, Sandisk, Cisco, entre outros. Sabe como é: assim como a internet, essa tal SAP realmente chegou para ficar.

8.1 SAP Netweaver Gateway

Se existe uma coisa frustrante no desenvolvimento ABAP puro é a sua incapacidade, como programador, de gerar uma experiência de usuário satisfatória. Os controles de tela disponíveis nativamente para o *SAP Gui* (cliente local para conexão e exibição de telas do SAP ERP) são extremamente limitados, incapazes de competir com tecnologias avançadas, como iOS, Android, HTML5 e derivados.

O maior mérito das telas da SAP Gui atuais é a velocidade, tanto de desenvolvimento quanto de uso – em relação ao fluxo de uso para preenchimento de campos, não à performance do programa como um todo. Gerar uma tela básica (conhecidas como *Dynpro* ou *Dynamic Programs*) no ABAP por meio de um Report, com 15 campos de filtro contendo ajudas de pesquisa complexas, é tarefa de poucos minutos.

O ABAP abusa de um dicionário de dados completamente integrado que facilita muito essa tarefa, por automaticamente disponibilizar uma série de funcionalidades ao criar um link entre o campo da tela e um campo de uma tabela do banco de dados. Toda essa praticidade tem um preço, que é a padronização das telas do sistema.

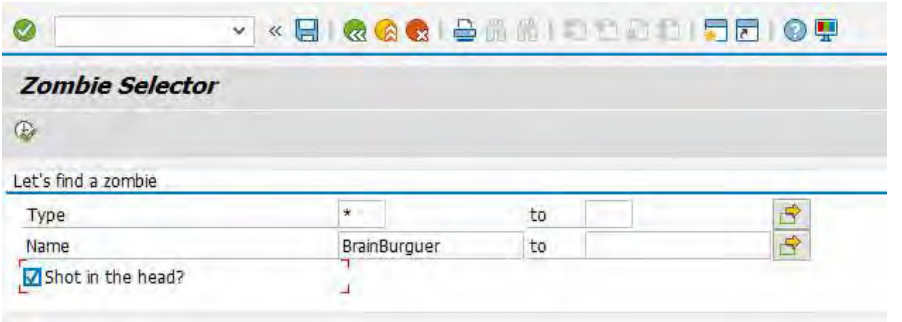


Fig. 8.1: Tela de seleção simples, muito comum dentro do sistema

Além das Dynpros, alguns objetos complementares surgiram para tentar enfrentar os problemas de usabilidade e limitação, como o Webdynpro e BSPs (*Business Server Pages*). Ambos permitem a criação de telas a partir do ABAP, para exibição em browsers.

No Webdynpro, a ferramenta de criação de telas web do ABAP gera o código automaticamente para ser exibido no browser; já no caso dos BSPs, é preciso codificar todas as interfaces na mão. O interessante é que mesmo sendo objetos disponíveis de forma nativa em qualquer ambiente SAP ERP, ambos não se tornaram opções populares, provavelmente por conta da praticidade de criação presente no uso das Dynpros. O usuário, por sua vez, já está acostumados com Dynpros (por conta das transações standard), e normalmente não questiona tal escolha.

Entretanto, não é mais uma simples comodidade ter de acessar sistemas empresariais através de interfaces simples e objetivas, isso é uma **necessidade**. O design de aplicativos para browsers ou sistemas operacionais tende a interfaces que ajudam usuários a completar tarefas com sucesso, normalmente com visual simples, e inteligência robusta e complexa por baixo dos panos.

Pense em como você fazia antigamente para comprar um ingresso de cinema: você precisava arranjar um jornal com os horários e dias, ir até o cinema, enfrentar uma fila enorme e descobrir na hora se existia ou não sessão para o filme que você quer ver. Hoje, com um smartphone, é possível realizar todo esse processo de qualquer lugar: desde a avaliação da disponibilidade de lugares na sessão até a compra efetiva dos ingressos. E nem precisa imprimi-

los, basta apresentar o ticket da compra na tela do celular e você entra no cinema. Como fazer para criar experiências semelhantes no mundo das empresas e seus ERPs?

Um dos caminhos seria a conversão total das telas atuais para telas mais modernas. Mas imagine o esforço necessário para adaptar todas elas (standards e customizadas). O esforço para algo assim seria completamente insano, já que a grande maioria das transações que os usuários utilizam está criada nas velhas Dynpros. Provavelmente, quando o trabalho estivesse concluído (se é que é possível terminá-lo), as "novas" telas já estariam defasadas, dada a quantidade monstruosa de tempo necessário para fazer tal adaptação.

Não seria mais fácil prover algum meio de aplicações externas acessarem os dados do ERP, de uma forma consistente e gerenciável, desenvolvendo a *User Interface* (UI) em outras tecnologias? A SAP permite que um cliente possa consumir dados do ERP a partir de qualquer tipo de aplicação, criando uma UI específica na tecnologia que bem entender. Algo assim entregaria a responsabilidade da criação dos fluxos das telas para sistemas externos ao SAP, removendo a necessidade do uso de Dynpros, Webdynpros, BSPs e toda essa velharia. É exatamente para tratar esse tipo de cenário que foi criado o SAP Netweaver Gateway, um *Add-on* (espécie de plugin) para o SAP ERP.

Consuma-me, diz o OData

Através do Gateway, desenvolvedores ABAP podem modelar um serviço OData (*Open Data Protocol*) para ser consumido por alguma fonte externa. Essa fonte externa pode ser absolutamente *qualquer* sistema ou tecnologia que consiga consumir um serviço OData, seja ele um aplicativo web, mobile, desktop ou uma máquina de café. Não é uma ferramenta exatamente nova, mas vem ganhando cada vez mais notoriedade nos tempos atuais, por motivos que explicarei na seção 8.3, junto ao SAP Fiori.

Protocolo OData

O OData é um protocolo aberto, criado pela Microsoft e utilizado como padrão pelo Gateway para qualquer tipo de comunicação, suportando operações de criação, leitura, deleção, atualização e consultas (*queries*). Talvez isso pareça algo de outro mundo para o ABAPer acostumado com o ALV (*SAP List Viewer*) de sempre, mas é basicamente a forma de comunicação entre front-end e back-end.

Exemplo: uma aplicação web faz uma requisição de leitura de ordens de venda para um serviço OData, criado pelo Gateway, que acaba resultando na chamada de um método ABAP no qual é feito o `SELECT` na tabela `VBAK`. Os dados são retornados por um parâmetro do método que cai na lógica do serviço OData, que, por sua vez, entrega os dados de volta ao aplicativo web para serem exibidos.

Caso você queira aprender com mais detalhes sobre serviços OData, o próprio site do protocolo é um bom início: http://www.odata.org/.

Não é preciso estar na última versão do ERP para poder usufruir do Gateway, a instalação é suportada desde a versão 7.0 do SAP Netweaver. Para saber quais componentes estão liberados para cada versão (a lista é muito extensa), recomendo o site de pré-requisitos do Gateway (http://bit.ly/2bRpVGu) , que é constantemente atualizado quanto ao suporte do Add-on, nas diversas versões do SAP ERP. Após sua instalação, os ABAPers terão acesso a um dashboard (transação SEGW), em que é possível modelar e monitorar os serviços OData.

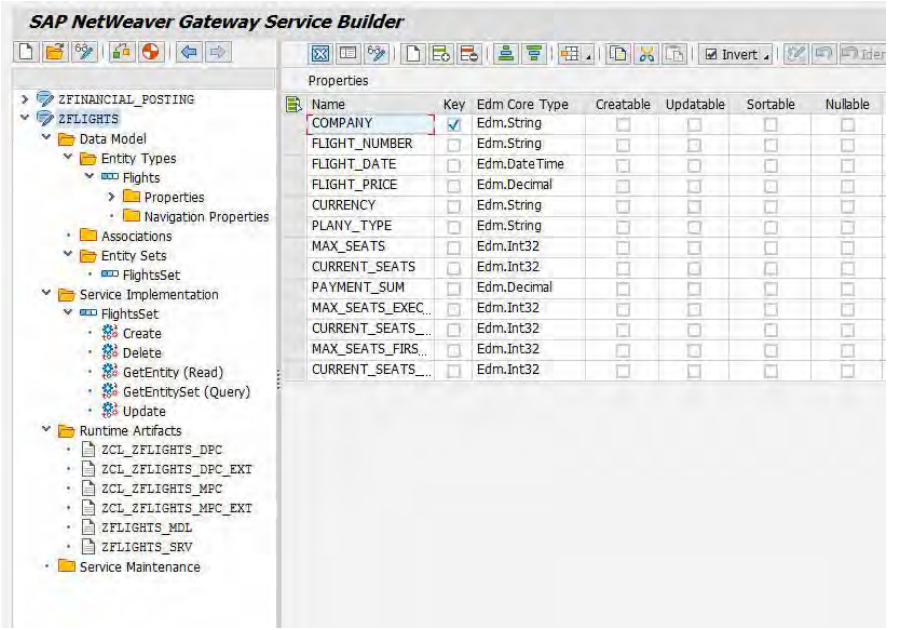


Fig. 8.2: Dashboard do Gateway onde os serviços são criados

Para entender o funcionamento completo do Gateway e como ele se comunica com as aplicações externas, é preciso estudar um pouco mais sobre o protocolo OData e arquitetura REST, que são as bases para toda troca de dados. Por hora, como ABAPer, é importante saber que existe uma ferramenta da SAP que permite que um sistema externo consulte, atualize, crie ou apague dados do ERP, fazendo chamadas para esses serviços OData que resultaram em chamadas de métodos, dentro de classes ABAP.

Nesse cenário, o ABAPer terá a função de desenvolvedor back-end, tratando os dados e criando lógicas consistentes, para que o aplicativo externo possa consumir a informação já trabalhada. Se você odeia desenhar telas e não entende nada de desenvolvimento front-end, o papel de desenvolvedor back-end pode ser um caminho a trilhar no futuro.

Arquitetura REST

O OData é um protocolo que se baseia na arquitetura REST (*Representational State Transfer*), que provê restrições, guidelines e melhores práticas para a criação de um serviço web. O REST não é o protocolo em si, é um modelo de arquitetura que pode ou não ser usado como base.

Como a arquitetura do OData tem raízes fortes na arquitetura REST, é essencial entender melhor como o REST guia a implementação e a comunicação entre as partes, para compreender completamente o funcionamento de um serviço OData. Já para a utilização básica do Gateway via ABAP, não é necessário ter conhecimentos profundos nesse tópico, já que há uma série de wizards e UIs que simplificam todo o processo. Entretanto, se você quiser se aprofundar mais nos conceitos, recomendo esta série de posts: http://bit.ly/1Gk4GqQ. Nela o autor guia o ABAPer na criação de uma API baseada em REST, usando somente o ABAP puro, sem o Gateway e serviços Odata. É uma boa forma de absorver os conceitos, para depois se aprofundar no protocolo OData em si.

Muitas siglas? Muitos nomes novos? Muita animação e confusão? Vamos ser um pouco mais práticos com um pequeno exemplo.

Chegando no ABAP a patir de uma chamada OData

Suponha que queremos fazer uma tela muito louca para buscar dados de um material da empresa, mas necessitamos de informações que estão disponíveis somente dentro do ERP. Precisamos que nossa aplicação web se conecte ao SAP, de alguma forma, e encontre esses dados, para podermos exibi-los dando piruetas virtuais.

Supondo que exista um serviço OData com o Gateway disponível no SAP ERP para busca de materiais, o aplicativo poderá consumir o serviço OData por uma URL (gerada pelo Gateway), como essa: http://www.zombie.com:50000/sap/opu/odata/sap/zflights_srv/FlightsCollection .

Essa chamada vai disparar uma chamada ao método de uma classe do

lado ABAP (também gerada pelo Gateway). Nesse método, o ABAPer pode fazer `SELECT`s, chamar `BAPI`s ou "marretar" tabelas para retornar os dados do material. O Gateway pega o retorno do método, gera um `XML` com os dados (ou um `JSON`, se preferir) – seguindo os padrões OData –, e envia para a aplicação que requisitou as informações.

Daí para a frente, a aplicação que consumiu o serviço OData pode fazer o que for preciso com os dados. Esse tipo de conexão não precisa ser necessariamente uma busca: pode ser uma ação de deleção, criação ou alteração.

Toda essa estrutura de URLs, métodos e disponibilização do OData é gerada automaticamente a partir do dashboard do Gateway, onde é possível modelar os serviços, e gerar todos os objetos ABAP e de configuração necessários.

Podemos afirmar, portanto, que o Gateway faz a conexão entre dois aplicativos ou sistemas diferentes. Esse conceito básico acaba confundindo o uso do Add-on com outro produto da SAP: o PI (*Process Integration*), *middleware* para gerenciamento de interfaces.

Porém, todas as features do Gateway focam em prover os dados do ERP de forma que possam ser consumidos e/ou manipulados por aplicação externas que envolvam **ações** do usuário. Já o PI tem uma abrangência muito maior, atuando como um ponto único para integrações de qualquer natureza, entre o SAP e sistemas externos. Apesar de parecerem similares, o posicionamento de cada ferramenta é bem diferente dentro do ecossistema.

Essa ideia de transformar o ERP como um plataforma capaz de expor dados de forma "consistente para o consumo" está diretamente amarrada com um novo produto do SAP, focado em telas. Vamos a ele.

8.2 SAPUI5 e OpenUI5

Com o Gateway provendo uma forma "padronizada" e "aberta" para o consumo de dados do ERP, é natural que a SAP tenha criado a sua própria biblioteca para criação de telas, chamada **SAPUI5**. Ela possui uma série de componentes que facilitam o desenvolvimento de aplicações web, abusando de HTML5, CSS3 e JavaScript.

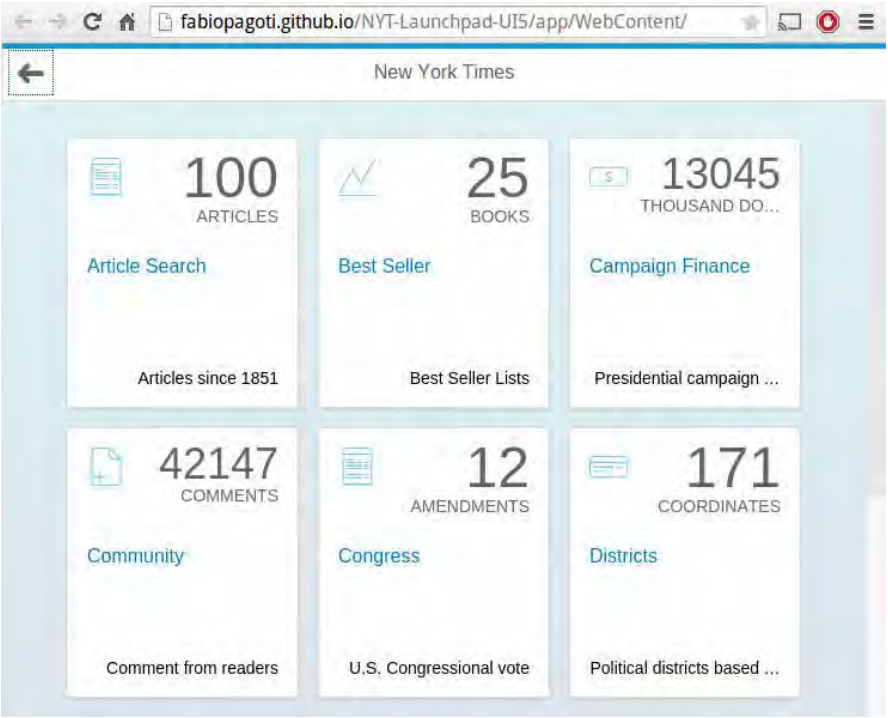


Fig. 8.3: Exemplo de um aplicativo feito com UI5

As vantagens do uso de uma biblioteca como essa é que diversos problemas que um desenvolvedor front-end enfrenta já foram sanados, sendo um dos mais complexos o suporte a diferentes tamanhos de tela e *devices* (por exemplo, browsers de computador, tablet ou celulares). Uma das principais features do SAPUI5 são seus controles que foram criados pensando no design responsivo.

Usando corretamente os controles disponíveis, é fácil criar telas com suporte a diferentes tamanhos de tela, sem precisar preocupar-se com as milhares adaptações de CSS ou JavaScript que seriam necessárias. A biblioteca resolve tudo sozinha.

É importante destacar aqui que a SAP não é, nem de longe, a primeira a criar uma biblioteca desse tipo. Há alternativas muito mais populares entre desenvolvedores web, como o JQuery, Zepto.js e Kendo UI, que já são apli-

cadas no mercado há anos. A grande diferença é que o SAPUI5 é a escolha nativa da SAP para usar em seus próprios produtos, o que naturalmente vai escalar para todo o ecossistema.

O uso do SAPUI5 está restrito aos clientes que contêm algum tipo de licença de produto SAP, mas isso não quer dizer que o seu uso precisar estar obrigatoriamente relacionado com um componente SAP. O SAPUI5 é realmente uma biblioteca voltada ao desenvolvimento web, com capacidade de gerar telas em qualquer tipo de aplicação web. A diferença é que o seu código e desenvolvimento são proprietários.

Você já deve estar pensando: "Puxa, mais uma vez os desenvolvedores do mundo SAP estão totalmente amarrados às licenças dos clientes... certo?" Não exatamente. Existe uma alternativa ao SAPUI5, chamada **OpenUI5**. A biblioteca é praticamente a mesma, porém o OpenUI5 é totalmente open source.

Sua criação ocorreu devido a uma pequena pressão da comunidade, que não via sentido em manter uma biblioteca com tanta empregabilidade atrelada aos clientes que possuem licenças SAP. Felizmente, a SAP resolveu abrir o código, criando uma *branch* específica para o mundo open source. Na minha opinião, esse movimento foi algo muito positivo para quebrar essa ideia de uma empresa extremamente fechada. Entretanto, se o OpenUI5 terá realmente alguma relevância no mundo web, só o tempo dirá.

Bem, muito bonitas essas telas e essa coisa toda de desenvolvimento web, open source e design responsivo. Mas o que será que a SAP fará com tudo isso? Fique tranquilo, ela já fez.

8.3 SAP Fiori

Sendo o Gateway uma plataforma robusta criada pela SAP para prover dados, para serem consumidos externamente, e o SAPUI5 uma biblioteca que permite a criação de telas web responsivas, o que a impediria de criar seus próprios aplicativos externos para exibir informações do ERP, de uma forma mais atrativa? A fórmula não poderia ser mais simples: Gateway + SAPUI5 = **SAP Fiori**.

A SAP mapeou quais transações standard eram mais usadas pelos usuá-

rios do ERP, reescreveu todas utilizando o combo SAPUI5 e Gateway, e disponibilizou-as dentro de um novo produto, o SAP Fiori. Essas transações são separadas em aplicativos dentro do pacote Fiori, podendo ser instaladas uma a uma como Add-ons no sistema, de acordo com a necessidade do cliente. Não existe nenhum custo adicional para um cliente do ERP utilizar o Fiori, basta ter o Gateway instalado no sistema para que "as novas transações" possam ser instaladas.

As telas são visualmente agradáveis, funcionam em diferentes resoluções, são otimizadas para dispositivos móveis e conseguem, finalmente, entregar aos usuários do ERP a flexibilidade e simplicidade que todos sempre sonhavam. Ok, sonhar talvez seja uma palavra muito forte, mas você pode ter uma ideia muito boa de como o Fiori funciona no site oficial: http://bit.ly/29sraxv.

Ei, há um capítulo atrás, este era um livro de ABAP, não era? O que Fiori, UI5 e Gateway têm a ver com ABAP?

Note que eu disse que a SAP reescreveu aplicações standard, não falei nada sobre transações customizadas com o ABAP. Se o cliente usar uma transação standard na versão Fiori, não vai demorar muito para ele perguntar se é possível converter uma de suas transações Z para esse novo formato. É ai que entra um desenvolvedor web, para criar as telas com o SAPUI5, e você, ABAPer, para criar a lógica de acesso aos dados com o Gateway.

As aplicações Fiori standard também podem ser ampliadas, permitindo alterações e ajustes (algo que lembra de longe um *user-exit*, ou brecha controlada para alterações). Qualquer aplicativo Fiori depende diretamente dos dados armazenados no ERP para funcionar, e o processamento desses dados irá fatalmente envolver o bom e velho ABAP. Entretanto, gradualmente mais e mais desenvolvedores ABAP vão passar a considerar o SAPUI5 como uma alternativa de exibição para seus aplicativos (seja por escolha própria ou por requisição do cliente).

Uma coisa engraçada que já ocorre é que as empresas/consultorias começaram a criar vagas para desenvolvedores "Fiori" quando, na verdade, tudo o que elas querem são desenvolvedores com conhecimentos em criação de programas com o SAPUI5. A única forma de conseguir um "desenvolvedor Fiori" é arrancando alguém que tenha criado um aplicativo Fiori na SAP e

contratando-o.

Parte dessa confusão se dá por conta do termo "Fiori-like", usado para descrever aplicativos SAPUI5 que se parecem com os aplicativos do pacote Fiori. Enfim, se precisei de tudo isso para explicar toda essa movimentação direcionada ao desenvolvimento web, imagine tentar convencer alguém sem noção nenhuma de tecnologia (como muitos gerentes, diretores e pessoas com poder de decisão) sobre essa diferença entre uma aplicação Fiori que é feita com SAPUI5 e um aplicativo criado com o SAPUI5 em si.

Mas a maior e mais complexa estratégia de expansão e consolidação da SAP não se resume a reescrever telas e invadir o mundo web. Ela tem a ver com um novo banco de dados.

8.4 HANA, HANA, HAHA... e mais HANA

O HANA é um banco de dados in-memory, criado pela SAP. Um produto completamente novo que visa aumentar drasticamente a performance do acesso e manipulação dos dados. Ele tem total aderência aos produtos atuais da SAP e é a base para praticamente tudo que a SAP está lançando nos últimos anos.

Antes de explorarmos um pouco o ecossistema criado a partir do HANA, vamos entender do que se trata esse banco de dados e por que ele é tão importante.

O poder de um banco de dados in-memory

O termo in-memory indica que o banco dados vai utilizar o armazenamento em memória de forma primária, diferente das versões de banco de dados mais usados hoje em dia no mercado SAP, que se baseiam primariamente no armazenamento em disco. A maior parte das instalações do SAP ERP em clientes está acoplada a bancos de dados de empresas externas, sendo mais comuns instalações com Oracle ou Microsoft SQL Server (ambos possuem suas próprias versões in-memory, mas as usadas em bases, instaladas do SAP ECC, costumam ser as versões convencionais).

O principal motivo dessa mudança é a **performance**, que é drasticamente melhorada em aplicações que lidam com bilhares de registros todos os dias.

Sendo o HANA uma criação da SAP, na evolução dos outros produtos sempre existirá algum tipo de integração nativa com esse novo banco de dados.

Antes de falarmos de integração, há uma mudança de arquitetura do HANA que é simples de ser explicada e ajuda bastante a entender como algumas mudanças "por baixo dos panos" beneficiam as aplicações. Essa mudança é o armazenamento dos dados de tabelas em coluna em vez do armazenamento em linhas.

Para entender melhor este conceito, vamos analisar inicialmente uma tabela que armazena seus dados em linhas:

*OV	Cliente	Status	Deleção
3201	'Casas Bahia'	Finalizado	(vazio)
3202	'Americanas'	Pendente	'X'
3203	'Casas Bahia'	Pendente	(vazio)
3204	'Wallmart'	Finalizado	(vazio)
3205	'Casas Bahia'	Pendente	'X'

Fig. 8.4: Exemplo de armazenamento em linhas

Se pegássemos os mesmo dados e migrássemos para o HANA, as informações seriam organizadas em colunas nas novas tabelas, de forma que nenhum dado fosse armazenado de forma duplicada:

*OV	Cliente	Status	Deleção
3201	'Americanas'	Finalizado	(vazio)
3202	'Casas Bahia'	Pendente	'X'
3203	'Wallmart'		
3204			
3205			

Fig. 8.5: Exemplo de armazenamento em colunas

Essa pequena mudança de arquitetura escala uma série de vantagens para o HANA:

- O espaço usado é otimizado, pois não existe informação duplicada em uma mesma coluna. Em produção, uma tabela de informações contábeis – como a `BSEG` – pode ter facilmente mais de 300 (!) colunas e

dezenas de milhões de registros. Imagine a compressão para uma tabela desse tamanho?

- Não existe a necessidade de índices secundários. Cada coluna já fica organizada com uma espécie de `SORT` nos valores armazenados. Isso quer dizer que acessos a quaisquer colunas serão performáticos.
- Operações analíticas serão extremamente mais rápidas. Um `SELECT SUM`, por exemplo, terá uma performance muito melhor em comparação aos bancos de dados convencionais, principalmente em tabelas gigantescas.

Essa ideia de armazenamento em colunas pode parecer estranha e talvez você esteja se perguntando: "Se o HANA vai armazenar as informações por colunas, como ele faz para montar uma linha?" Internamente, cada uma das tabelas contém uma tabela auxiliar com os índices de cada coluna que formam as linhas. Para o exemplo anterior, a tabela de índices ficaria assim:

*Registro	Índice das Colunas
1	1,2,1,1
2	2,1,2,2
3	3,2,2,1
4	4,3,1,1
5	5,2,2,2

Fig. 8.6: Tabela de índices que permite montar as linhas do exemplo anterior

É claro que existe um custo para "recriar" uma linha a partir dessa tabela de índices, mas esse formato de tabelas em colunas beneficia operações com grandes quantidades de dados. Qualquer guideline de performance ABAP pré-HANA dirá que é muito melhor buscar tudo o que você for precisar do banco de dados com um ou mais `SELECT`s, e trabalhar os cálculos na memória interna do programa.

Com o HANA, o cenário muda, pois é mais performático delegar os cálculos e agregações complexas para o banco e trazer a informação já trabalhada para o programa, sobrando somente o trabalho de exibição. A SAP

usa muito o termo *code-to-data* para exemplificar essa ideia que vem com o HANA: cálculos complexos podem ser feitos diretamente nas queries executadas no banco em vez de serem feitos nos programas.

Essa arquitetura impacta ainda a distribuição da informação dentro do SAP ERP. Considere as famosas "tabelas espelho", que nada mais são do que a duplicação de uma tabela com uma chave primária diferente. A necessidade de duplicar dados se dá por conta do tamanho enorme da tabela principal, que inviabiliza operações complexas por conta da má performance de acesso. Com as tabelas espelho, o sistema reduz as informações e as deixa organizadas por "visões" diferentes de negócio (na prática, as tabelas principais costumam ter tantos registros que as espelhos também acabam virando tabelas "monstrinhas").

Um dos casos mais conhecidos entre ABAPers é o combo de tabelas `BSEG`, `BSIK`, `BSAK`, `BSID` e `BSAD`, que armazenam basicamente a mesma informação contábil em estados diferentes para o negócio (documentos compensados e não compensados, por cliente ou fornecedores). No HANA, essa complexidade toda acaba, pois juntando a compressão com a performance de acesso a grandes massas de dados, as cinco tabelas tornam-se uma. Isso permite a realização de qualquer análise necessária, com alta performance.

Levando em consideração somente a prática ABAPer, o HANA exige que a forma de pensar no tratamento de dados em um programa seja completamente diferente. Se um ambiente migrar para o HANA, mas manter os seus códigos priorizando a aplicação de cálculos e agregações em programas – em vez de delegar essa atividade para o banco –, o ganho de performance será irrisório e o HANA será subutilizado.

A SAP provê uma série de facilidades para mitigar essas dificuldades em uma migração, entretanto, essa é uma mudança de pensamento que demorará para ser incorporada em larga escala pelos ABAPers. A coisa toda fica ainda mais demorada quando considerarmos que fazer a migração de banco de dados é uma tarefa extremamente complexa em grandes empresas, envolvendo diversas áreas e profissionais.

Tudo isso que acabei de escrever vai demorar muito para ver a luz do dia, certo? Errado, pelo simples fato de o HANA **não** precisar do SAP ERP para *existir* como produto.

Ecossistema

O HANA pode ter nascido como um banco de dados, mas hoje a palavra HANA representa muito mais do que isso. Em vez de ser tratado pela SAP "somente" como um banco de dados in-memory, ele é tratado como uma *plataforma* in-memory.

Através do *SAP HANA Extended Application Services* (XS), um desenvolvedor consegue criar e executar a parte do servidor de uma aplicação (com JavaScript), delegando a atividade de exibição para outras tecnologias.

Assim como o Gateway, o XS consegue prover um serviço OData a partir do HANA, para ser consumido por tecnologias front-end. Isso quer dizer que é possível fazer um aplicativo usando o SAPUI5, e conectá-lo diretamente ao HANA, sem a necessidade de "passar" pelo ABAP. Isso não se restringe ao SAPUI5 ou OpenUI5, mas a qualquer tipo de biblioteca, framework ou linguagem que consiga usar o protocolo OData para comunicação com o back-end.

É possível também usar o HANA na nuvem, por meio da *HANA Cloud Platform* (HCP). Um desenvolvedor pode criar um ambiente *trial free* usando um sistema como o *Amazon Web Services* (AWS) e criar seu aplicativo na plataforma, para posteriormente ser vendido através do *HANA App Center* (http://www.sapappcenter.com/) .

É interessante ver que há algumas startups ofertando aplicativos HANA no App Center. Existe uma iniciativa da SAP para expandir o uso do HANA por startups, por meio do programa *Startup Focus* (http://startupfocus.saphana.com/) .

Toda a expansão do HANA é algo completamente diferente do que a SAP fez nos últimos 30 anos, sendo até um pouco difícil de ser entendida pelos mais experientes da área, dada a avalanche de informações existentes sobre o tema. No Brasil, ainda não há uma penetração imensa do HANA como banco de dados ou plataforma nas empresas, mas aos poucos isso está mudando, principalmente por conta de um produto chamado *Tax Development Framework* (TDF).

O TDF é uma plataforma para desenvolver programas que geram relatórios de obrigações fiscais (como os SPEDs) feita em HANA. Em vez de gerar os relatórios somente por programas ABAP, a empresa instala o TDF, migra os dados para o HANA e gera os relatórios diretamente de lá, com dashboards

criados por parceiras da SAP (que podem ser feitos em ABAP, ou em outras tecnologias).

Nessa história de usar o HANA como plataforma e expandir para diversos outros produtos, como fica o SAP ERP? Lá atrás, no capítulo 2, expliquei rapidamente que o SAP ERP, no qual a maioria das ABAPers trabalham hoje em dia, é uma evolução do SAP R/3, com sua arquitetura em 3 camadas.

Claramente a SAP está pautando o futuro no HANA, portanto, é natural que o ERP também migrasse aos poucos para a plataforma. Em fevereiro de 2015, a SAP anunciou o próximo passo do seu maior produto, o SAP ERP, com o chamado S/4HANA (algo como "Simple for HANA". Não, não é SAP, e não, eu não sei porque deixaram aquela barra no nome).

No futuro que a empresa alemã imagina, todos os ERPs da SAP vão rodar em cima da plataforma HANA. Se ela já tem uma aplicação de sucesso com mais de 30 anos de estrada e agora também tem um banco de dados promissor, nada mais justo que unir os dois em uma coisa só. Qualquer um de nós faria o mesmo.

8.5 A posição do ABAP no "novo mundo"

Em 2007, um livro chamado *Next Generation ABAP Development* foi lançado pela dupla Rich Heilman e Thomas Jung. A ideia por trás desse livro era a de demonstrar um panorama geral das features do ABAP que, naquela época, eram subutilizadas em empresas e poderiam trazer benefícios a vida dos ABAPers.

O livro trata de assuntos pertinentes para a data em que foi lançado, como BSPs, Web Dynpro, Portal, Webservices, classes de persistências, objetos compartilhados (*Shared Objects*) e até mesmo ABAP Unit (ferramenta para aplicar o Test Driven Development no ABAP). Ou seja, todas features que continuam sendo subutilizadas por ABAPers mundo afora.

Escrevi no capítulo 7 uma breve história que demonstra como o posicionamento de alguns ABAPers acabam impedindo que inovações da linguagem sejam, de fato, aplicadas no dia a dia. Essa postura está fadada a um caminho de trevas, já que cada vez mais o ABAP tende a ser uma linguagem que vai *coexistir* com outras tecnologias da SAP, não sendo somente ela a protagonista.

Vejamos o exemplo da integração do SAPUI5 com o ABAP através do Gateway. Não é mais um questão de decidir se você vai usar uma classe ou uma função para resolver um problema, ou se criará mais um botão, em uma barra de `STATUS-GUI`, para criar uma nova ação. Para sequer criar uma tela, é preciso ter algum conhecimento de JavaScript, saber como um OData (criado via Gateway) funciona e ter noções boas de ABAP Objects, para poder utilizar corretamente as classes geradas para implementar a lógica funcional no back-end.

Supondo que o cliente não ligue muito para a implementação de novas tecnologias front-end a curto prazo, ou mesmo que esse tipo de desenvolvimento fique dividido (ABAPer no back-end, desenvolvedor web no front-end), há features criadas na mais nova versão do ABAP (7.40) que integram ainda mais a linguagem aos bancos de dado modernos, com grande foco na integração ABAP e HANA:

- **ABAP Managed Database Procedures:** com a nova versão, é possível criar *procedures* diretamente de dentro do ABAP, por meio de um método de uma classe. Basicamente, é só adicionar uma extensão na chamada do método e proporcionar mais algumas configurações, por exemplo, `METHOD xxxxx BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT`. Neste caso, `HDB` indica que o banco de dados é o HANA, e `SQLSCRIPT` é a linguagem usada nele para criar procedures. Dentro desse métodо, a programação não é feita em ABAP, mas sim na linguagem enviada pelo parâmetro `LANGUAGE`.

- **ABAP Core Data Services:** a ideia por trás do CDS é permitir a criação de scripts que vão gerar *views* muito mais complexas do que aquelas que poderiam ser criadas pela SE11, para serem consumidos por meio de um `SELECT`. Esses scripts podem ser reutilizados por diferentes bancos de dados.

- **Evolução do OpenSQL:** o conjunto de instruções, usados pelos ABAPers para fazer seleções, chamado de OpenSQL, evolui para permitir que mais lógicas sejam resolvidas pelo banco de dados. Há muitas possibilidades novas, como a utilização de instruções como o `CASE`, para

decidir os valores das colunas na tabela de retorno, cálculos entre colunas dentro do próprio `SELECT`, ou mesmo a concatenação de valores entre colunas.

Todas essas features estão alinhadas com o paradigma *code-to-data*, em que a ideia é fazer com que mais código seja resolvido na camada do banco de dados em vez de simplesmente buscar todos os dados com milhares de `SELECT`s e tratar a junção de dados de diversas tabelas via ABAP.

> A lista completa das novidades pode ser acessada em
>
> http://bit.ly/2bPEqJX

Além das inovações que focam na maior integração com bancos de dados, há muitas outras criadas com o objetivo de modernizar a forma como a linguagem é escrita, como por exemplo o operador `NEW` para instanciar um objeto, ou as declarações *in-line* que permitem que um `FIELD-SYMBOL` seja criado diretamente dentro de um `LOOP`, como `LOOP AT <tabela> ASSIGNING FIELD-SYMBOL(<linha>).` São melhorias que reduzirão drasticamente a quantidade de código necessária para realizar determinada tarefa.

Um parte interessante dessas novidades é que muitas delas só estarão disponíveis para os ABAPers que programarem utilizando o *ABAP Development Tools* (ADT), um *plugin* do Eclipse (software para desenvolvimento) que habilita a programação ABAP fora da SAPGui. CDS views ou procedures via ABAP só poderão ser criadas com o Eclipse, e não estarão disponíveis para criação pela boa e velha transação SE80. Com o Eclipse também deixa de existir a dependência de utilizar o Windows como plataforma de desenvolvimento, já que ele funciona perfeitamente no Linux ou no MacOS.

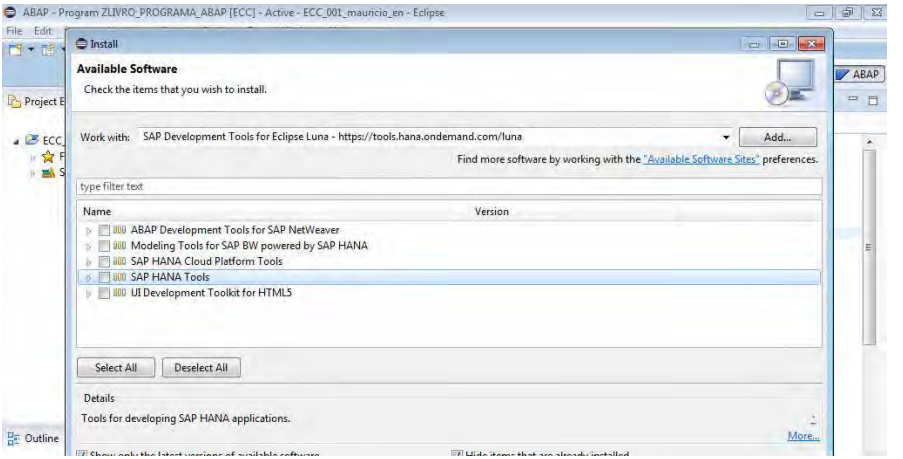


Fig. 8.7: Outra vantagem do Eclipse é que, além do ABAP, ele também suporta desenvolvimentos em UI5 e na plataforma HANA

Quebrando barreiras

Há um novo livro que explora com muito mais detalhes todas essas novas ferramentas e features chamado *ABAP to the Future*, escrito por Paul Hardy. A ideia é parecida com a do *The Next Generation ABAP*: explorar tudo o que já está disponível e é subutilizado pela comunidade ABAP, com casos de uso e exemplos de implantação.

Entretanto, o que garante que a comunidade não vá, em sua maioria, novamente ignorar essas novas ferramentas, fazendo com que muitas dessas boas ideias caiam no esquecimento?

É impossível prever o futuro, mas o posicionamento da SAP em usar o HANA como base do seu novo ERP tende a exigir que o desenvolvedor ABAP domine muito mais tecnologias, além do velho `SELECT` no `LOOP`. O desenvolvedor ABAP precisará entender que a linguagem passa a coexistir com outros produtos, e deverá abusar de todas as novas features para potencializar melhores soluções para os usuários.

A maior barreira a ser quebrada por profissionais desta área é a de não se apegar aos problemas discutidos ao longo deste livro. São vícios e formas de pensar criadas ao longo do tempo, que prejudicam completamente uma mu-

dança de paradigma e da forma como o ABAPer pode encarar seu trabalho.

Atrás dessa barreira, há um mundo com muitas ferramentas novas, formas diferentes de trabalho a serem exploradas, tecnologias que podem gerar experiências fantásticas e oportunidades de carreira (quase) infinitas.

O ABAP *sempre* esteve em constante evolução. Não se apegue ao passado, mas evolua com ele!

Capítulo 9

Apêndice I – Aprendendo ABAP por conta própria

Se você chegou até aqui lendo todo o resto do livro, há grandes chances de você já trabalhar com ABAP, fazendo, assim, todo o conteúdo deste apêndice desnecessário. Mas, para você que leu este livro pretendendo ingressar na área, saiba que existe uma forma de aprender ABAP sem precisar desembolsar litros de dinheiro.

Aprender sem depender de um curso pago ou de empresas é possível, mas está longe de ser uma alternativa fácil.

A maior diferença entre aprender Ruby, Java, C# ou ABAP, está na dificuldade de conseguir um sistema ERP para realizar o seu aprendizado. Sem um sistema Trial disponibilizado pela SAP, ou sem o acesso a um ERP de alguma empresa, você não terá como criar nem mesmo o famoso "Hello World" (mais

sobre isso ao longo do apêndice).

Essa dependência gera uma série de dificuldades que certamente podem ser superadas, mas precisam de uma certa dose de persistência por parte de quem quer aprender.

Outra coisa que complica a vida dos iniciantes é entender que o ABAP é criado em função das necessidades do ERP. No dia a dia ABAPer, é muito comum passarmos horas tentando arquitetar lógicas com regras de negócio complexas que integram e envolvem muitas tabelas preexistentes no sistema. Por conta disso, é preciso se contextualizar um pouco para entender onde você está "pisando".

ABAPers especialistas contêm um conhecimento bom de como os dados transitam funcionalmente entre as tabelas do ERP, o que facilita muito o trabalho. Na programação ABAP, conhecer diversas técnicas de programação e comandos mirabolantes realmente ajuda, mas entender como a informação transita dentro do sistema é tão importante quanto. É um desafio constante para o desenvolvedor ter esse tipo de conhecimento, mas faz parte da diversão.

Enfim, chega de história! Vamos começar?

9.1 Aprenda inglês

Esperava algo técnico? Sei que já citei isso no capítulo 1, mas saiba que esta é realmente a dica mais importante deste livro e algo que faz uma falta tremenda no cotidiano.

Aprender inglês é primordial para trabalhar com ABAP, SAP ou qualquer outra coisa relacionada ao mundo de TI. Para demonstrar isso na prática, adianto que você não conseguirá absorver nem metade das coisas que estiverem listadas neste capítulo se não souber inglês. Vou deixar um pequeno aviso nessas partes, para você saber o que estará perdendo.

Ter de criar uma sessão só para isso pode parecer loucura para algumas pessoas, mas praticamente todos que cruzaram o meu caminho profissional e não entendem inglês concordam que isso já fez **muita** falta em diversas ocasiões.

Eu poderia dedicar milhões de palavras sobre o quanto o inglês é impor-

tante para aprender qualquer coisa de tecnologia, mas decidi resumir: se você não quiser aprender inglês de verdade – e não só pseudoleitura –, a escolha é sua. Entretanto, tenha *certeza* de que você será um profissional de TI limitado. Talvez algum dia exista informação suficiente em português (já lançaram até uma versão em PT-BR do *StackOverflow*), mas isso não acontecerá tão cedo. Diariamente encontro pessoas incapazes de acompanhar os novos lançamentos e atualizar seus conhecimentos por conta disso.

It's up to you!

9.2 Entenda bem o que SAP ERP e ABAP querem dizer

Eu não estava brincando quando disse que o contexto é extremamente importante no aprendizado ABAP. Mas acalme-se! Sabe a Wikipédia? Acesse-a **em inglês** e procure pelas seguintes informações:

- SAP? Que empresa é essa? – http://en.wikipedia.org/wiki/SAP_SE
- Como funciona o ERP da SAP – http://en.wikipedia.org/wiki/SAP_ERP
- Overview do ABAP – http://en.wikipedia.org/wiki/ABAP

Há muitos livros e guias na web que tentam explicar o funcionamento básico do SAP ERP que podem completar o seu entendimento, mas os artigos da Wikipédia em inglês mantêm-se atualizados e servem para dar uma boa base aos iniciantes. Só tente não se perder no mar de links, pois existem muitos produtos SAP, e isso pode gerar uma certa confusão. Em pouco tempo, você entenderá que "mundo SAP" é sinônimo de "mundo das siglas".

ABAP é uma linguagem de programação proprietária da SAP, e tudo que você desenvolver em ABAP será para produtos da própria SAP. É diferente de linguagens abertas como Java ou C++, que podem ser usadas para desenvolver praticamente qualquer tipo de software que lhe der na telha.

É costume de desenvolvedor de software querer aprender a linguagem sem se importar com a história da coisa toda. No entanto, é muito fácil se

perder no meio de tanta informação cruzada que existe no mundo SAP. Compreender um pouco melhor a história do ecossistema facilitará muito o processo, principalmente na hora de entender as funcionalidades do ABAP (se você burlou o sistema e veio direto neste apêndice, os capítulos 1 e 2 podem ajudar com isso).

9.3 SAP em casa: criando sua estação de trabalho

A liberação de uma versão *trial* por parte da SAP já passou por diversas mudanças. No passado, você preparava o ambiente com uma lista bem chata de pré-requisitos, baixava um arquivo `.zip` gigante de instalação (o famoso MiniSAP), descompactava, rodava o instalador e torcia muito (**muito** mesmo) para que tudo desse certo.

Mesmo seguindo todos os passos dos guias da internet e preenchendo todos os pré-requisitos, algumas vezes a instalação falhava sem mais, nem menos. Eu mesmo já rodei a instalação duas vezes em uma mesma máquina virtual Windows: na primeira a instalação falhou, na segunda não. E já que desgraça pouca é bobagem, o processo todo demorava, em média, cerca de 10 horas. Bons tempos.

Pelo bem de sua sanidade (e de todos os que já são ABAPers), uma boa notícia: hoje em dia o processo é muito mais fácil. A SAP disponibiliza um acesso a um servidor trial sem custo para o desenvolvedor, por intermédio do *Amazon Web Services* (AWS). Acessando o site de desenvolvedores SAP – disponível em http://developers.sap.com/ –, você encontra links e guias que vão ensinar como fazer para ter a sua própria instância com um servidor ABAP Trial na nuvem.

Muito mais fácil que o método antigo, certo? Mas nem tudo são flores. O processo de criação de uma instância ABAP na AWS é cheio de detalhes e pequenas configurações que podem fazer você desistir no meio do caminho, principalmente se for leigo no assunto.

Entretanto, graças ao maravilhoso mundo dos blogs, há um guia extremamente detalhado disponível para você seguir, facilitando ainda mais o processo: http://bit.ly/2byZs1A (adivinhe em qual idioma ele

está escrito).

Note também que não existe custo por parte da SAP pelo servidor ABAP Trial, mas existem custos relacionados ao servidor na nuvem. Ter uma instância ABAP Trial na AWS envolve espaço de armazenamento nos servidores da Amazon, custos relacionados ao tráfego de informações e tempo de utilização da máquina.

Os valores estão longe de serem abusivos (normalmente frações de dólares por hora de utilização), mas é extremamente importante que você controle a instância manualmente. Como os valores são cobrados por hora, lembre-se sempre de manter a instância desligada quando não estiver usando o servidor para minimizar os custos.

Existem guias no portal de desenvolvedores para que você crie um alerta na AWS quando o valor exceder uma determinada quantia, por exemplo: "alerte-me quando a conta passar de 10USD/mês". Recomendo que você ative esses alertas para não ter surpresas quando chegar a conta.

Além do AS Trial para o ABAP, a SAP oferta uma série de ambientes diferentes que podem ser utilizados por desenvolvedores para listar as tecnologias SAP. Este link lista todos os ambientes disponíveis: http://bit.ly/1uHOjzF.

Se você tiver curiosidade sobre do que se tratam algum deles, há uma palestra em português feita pelo Tobias Hoffman, em um evento SAP brasileiro, que ajuda a entender um pouco mais sobre algumas delas: bit.ly/opcoes-trial-para-devs.

9.4 Aprenda a programar o básico

O maior problema da galera que vai começar no mundo SAP é achar que aparecerá "do nada" um PDF mágico com centenas de páginas, códigos explicados, respostas para todas as dúvidas e explicações com o máximo de detalhes. Vamos aos fatos: ABAP é uma linguagem usada para um software de uma *empresa* alemã, que é utilizado por *empresas* do mundo todo, em um mercado muito fechado. Você acha mesmo que as pessoas iriam sair explicando todos os detalhes, assim, de graça?

Apesar de muitas pessoas da área acreditarem que essas informações não são compartilhadas sem custo, a própria SAP disponibiliza um guia detalhado

sobre o aprendizado e uso do ABAP no SCN (http://bit.ly/2bDx2Ff) . Esse não é um curso oficial, é um documento que agrupa diversos artigos gerados pela comunidade que ajudam no ensinamento do ABAP.

Arrisco dizer que, ao final da leitura dos guias, você saberá de alguns conceitos e técnicas que muitos ABAPers velhos de guerra não conhecem. É importante notar que esses guias foram criados há alguns anos – entre 2007 e 2010 –, mas continuam com informações relevantes e úteis até hoje. A única parte que deve ser considerada é a da instalação do ambiente Trial, pelos motivos que expliquei em um tópico anterior (mas vale como história de como nós sofríamos!). Você conseguirá recriar todo o resto no seu ambiente Trial tranquilamente.

Além deste guia agregado de explicações, vários tutorias, livros impressos e *e-learnings* bacanas estão disponíveis na web para qualquer um que queira aprender. Uma rápida busca no Google retornará diversos lugares para aprender ABAP, inclusive sites brasileiros, como o ABAPZombie (http://www.abapzombie.com) e o ABAP101 (http://www.abap101.com) .

Se você seguir tudo, tenho certeza de que terá uma boa base de como o ABAP funciona, estando pronto para encarar o primeiro projeto.

Ah, e está tudo em inglês.

Fui de Noob para Beginner. Quero virar Intermediate, #comofaz?

Calma lá! Antes de querer atingir o nível *Intermediate* por conta própria, aprenda mais algumas coisas utilizando esses e-learnings (todos gratuitos):

- ABAP OO Tutorial (http://bit.ly/2bi8HFQ)
- Introdução a Web Dynpro (http://bit.ly/2bIjpUd)
- ABAP Debugger para Iniciantes (http://bit.ly/2bkSDgh)
- Editor ABAP (http://bit.ly/2bi917h)

Alguns destes links são para a Parte 1 do e-learning, as outras você encontra ou em links relacionados, ou com uma busca rápida no SCN. Todos focam em conceitos bem básicos e têm um formato bem didático de ensino, facilitando a absorção do conhecimento por um iniciante.

Caso você queira aprofundar seus conhecimentos em outras tecnologias que envolvem desenvolvimento ABAP – como UI5 e HANA –, há também o site da OpenSAP (http://open.sap.com/) , que oferece cursos gratuitos online.

Ah, e está tudo em inglês. De novo.

9.5 O maravilhoso mundo dos livros que ensinam tudo e mais um pouco

Como em qualquer outra linguagem de programação, existe uma grande quantidade de livros que explicam a linguagem em ABAP, alguns deles nos mínimos detalhes. Tem coisas explicadas em alguns livros que até quem criou o ABAP não deve saber (sério).

A SAP tem uma editora própria, a SAP Press, que publica livros sobre diversos temas. Minha recomendação principal é o livro *ABAP Objects*. Você vai aprender a programar desde o básico, utilizando Orientação a Objetos, e o nível de detalhes das explicações é extremamente alto, sem ser chato (para um livro técnico).

Para se aprofundar um pouco mais em outras ferramentas da linguagem, recomendo os livros *Next Generation ABAP* – de Rich Heilman e Thomas Jung – e *ABAP to the Future* – de Paul Hardy –, duas fontes enormes de conhecimento.

Há uma movimentação recente em torno do novo banco de dados da SAP (o HANA), e existe um livro chamado *ABAP Development for SAP HANA* – escrito por Thorsten Schneider, Eric Westenberger e Hermann Gahm –, que explica como o ABAP integra-se ao HANA. Não recomendo este livro de início, mas, pelos motivos explicados no capítulo 8, é importante saber que ele existe.

Pesquisou, viu os preços e achou um absurdo? Essa é uma reclamação constante, já que não existe publicações locais no Brasil e é preciso importar os títulos (dá para comprar um e-book também, direto no site da SAP Press. Pelo menos você economiza no frete). Livros assim são um investimento animal; afinal, você está comprando conhecimento, algo que muita gente da área ainda tem medo de compartilhar. Aposto que um curso oficial na SAP, ou extraoficial brasileiro sobre qualquer um desses temas, sairia muito mais caro.

Ah, e você já deve ter entendido o mantra do inglês até aqui, certo?

Mãos à obra!

Como eu disse inicialmente, este é um caminho bem árduo. Longe de ser impossível, mas cheio de nuances que podem fazer você desistir no meio do processo.

Se você for um dos brasileiros que não desistem nunca, todo conhecimento adquirido poderá ser muito bem empregado neste vasto mundo SAP.

Para terminar, deixo a seguir guia um pouco mais detalhado de como criar a instância ABAP Trial, utilizando a AWS. Suba a sua instância, abra o ambiente e mãos à obra!

Capítulo 10

Apêndice II – Links e indicações

10.1 Publicações, sites, guias, artigos e livros citados

Para que você não precise procurar ao longo do livro quando quiser visitar um link que citei sobre um determinar assunto, criei este compilado organizado por tópicos.

Para os iniciantes

- http://bit.ly/2biqp6f – Guia original do "ABAP Noob", com comentários relatando experiências de ingresso na área;
- http://bit.ly/abap-trial-guide-paul-hardy – Guia para instalação do ABAP AS Trial, por Paul Hardy;

Livros

- *Next Generation ABAP*, por Rich Heilman e Thomas Jung: explica diversas ferramentas interessantes da linguagem ABAP, da época em que foi lançado;
- *ABAP to the Future*, por Paul Hardy: evolução do *Next Generation ABAP*, porém com explicações para features mais atuais;
- *ABAP Developmento for SAP HANA*, por Thorsten Schneider, Eric Westenberger e Hermann Gahm: tudo que muda no desenvolvimento ABAP junto ao HANA;
- *ABAP Objects*, por Horst Keller e Sascha Krüger: aprendendo ABAP do zero através do paradigma de Orientação a Objetos;
- *Software Estimation: Desmystifying the Black Art*, por Steve McConnel: um livro inteiro criado com o intuito de ajudar um desenvolvedor a estimar o tempo gasto na criação de um código;
- *Clean Code: A Handbook of Agile Software Craftsmanship*, por Robert C. Martin: princípios, padrões e práticas para criar um código "limpo", como casos de usos reais de aplicação;
- *The Design of Everyday Things*, por Donald Norman: explora os conceitos básicos do design de interação, que visa melhorar a comunicação entre objeto e usuário.

10.2 Recomendações adicionais

Outros sites que recomendo fora do SCN, com conteúdo relacionado a SAP e ABAP:

- http://zevolving.com/ – Um site com muita informação relevante, principalmente para quem gosta de usar ABAP Object. O autor do site, Naimesh Patel, fez um série muito interessante onde implementou vários Design Patterns usando ABAP, com exemplos de utilização.

- http://abapinho.com/ – Blog de Nuno Godinho, de Portugal. Ele mantém uma frequência ótima de publicação desde o início do blog, em 2009. O maior foco são pequenas curiosidades e dicas para ajudar no dia a dia. E o site ganha, de muito longe, como melhor layout já feito em site de ABAP. :)

- http://www.hanabrasil.com.br/ – Um projeto de Fábio Pagoti (abap101) e Fábio Ferri. É um site novo com foco principal em desenvolvimento HANA, UI5 e derivados do novo mundo.

- https://github.com/fabiopagoti/NYT-Launchpad-UI5 – Por favor no Fábio, o aplicativo UI5 do New York Times, exibido no capítulo 8, que foi feito por ele e está disponível neste repositório.

- http://bit.ly/2bIMvSc – Canal do YouTube que agrega diversas palestras feitas em eventos brasileiros da comunidade SAP. Note que não são específicos de ABAP, são relacionados a diversos assunto do mundo SAP.

No capítulo 8, apresentei algumas ferramentas da "nova guarda" do mundo SAP. Mais alguns livros publicados sobre o assunto que valem ser mencionados:

- *UI5 para desenvolvedores ABAP*, por Fábio Pagoti do ABAP101 e HANABRASIL: foca em ensinar sobre UI5 e o mundo do desenvolvimento web para o ABAPer, com exemplos que aproximem os dois mundos.

- *OData and SAP Netweaver Gateway*, por Carsten Bönnen, Volker Drees, André Fischer, Ludwig Heinz e Karsten Strothmann: particularmente acho o Gateway uma ferramenta fantástica, e esse livro faz um ótimo trabalho em explicar toda a avalanche de conceitos e siglas que fazem parte do produto e do protocolo OData.

- *Getting started with SAPUI5*, por Miroslav Antolovic: um livro que foca somente na biblioteca da SAP para UIs, o SAPUI5, com bastante detalhes.

Para terminar, muitos ABAPers ainda não sabem direito como fazer para trabalhar com o ABAP no Eclipse. O guia, disponível em http://bit.ly/2bkzbFe, é um bom ponto de partida. Mas tenha em mente que cer-tas features só vão funcionar em ambientes que estejam com versões atua-lizadas (como sempre). Para saber o que funciona em qual versão, acesse: http://bit.ly/2bCbbJO.

Índice Remissivo