Ensaio sobre a conversão de um Projeto Clipper para xHarbour

Christian Linhares Peixoto

Introdução

"Em primeiro lugar, porque é mais barato; em segundo lugar, traz impressa na capa, em letras garrafais e amigáveis, a frase NÃO ENTRE EM PÂNICO". Douglas Adams

A empresa na qual trabalho tem um grande cliente no setor de comunicações que solicitou uma mudança tecnológica. O APLICATIVO que capta os dados das suas unidades de atendimento, ativo desde 1998, foi desenvolvido em Clipper 5.2 e atende tanto às unidades próprias quanto às terceirizadas. É dividido em 2 módulos: um realiza as operações de tesouraria da unidade de atendimento e o outro executa o atendimento em si, como venda de produtos, serviços, recebimento e pagamentos e emissão de tíquetes. Como era previsto a viabilidade de converter o sistema, iniciamos os testes do Harbour desde 2001 e o xHarbour desde 2003, sentimo-nos à vontade para realizar a mudança.

O Clipper é tido por muitos como uma linguagem antiga e de valor zero. Isto deve-se a dois fatores:

a) a Computer Associates, antiga proprietária, não levou adiante a ferramenta(o Visual Objects não conseguia converter os programas antigos de forma razoável) e

b) a maioria dos desenvolvedores sempre a utilizaram para acesso a banco de dados (DBF), raramente usando as vantagens da OO (Orientação a Objetos - disponível desde a versão 5.0.1), funções externas em C (também desde a versão 5.0.1) e aspecto visual 3D (implementado oficialmente na versão 5.3).

O xHarbour nasceu como um projeto baseado no Harbour, cujo objetivo é ser um “Clipper opensource”. A necessidade dos criadores do xHarbour era ter um produto xBase num prazo curto e que pudesse oferecer mais do que apenas compatibilidade com o Clipper original[1]. De forma simplista, este compilador converte os fontes PRG em C e um compilador C/C++ transforma-os em executável (utilizamos o Borland C++ 5.5 para Windows e o GNU gcc para Linux).

O APLICATIVO aproveitou muitas vantagens de OO, de C e de acesso a DBF(índices CDX, menores e mais rápidos), mas ficou à mercê do Clipper. Como há interesse na continuidade do APLICATIVO para uso em outros tipos de unidades de atendimento, incluindo a exportação deste software, a gerência aprovou a conversão do Clipper para xHarbour, especialmente por não ter custos adicionais. Para que o projeto fosse um sucesso frente ao escopo e prazo previstos, utilizamos um método de desenvolvimento simples e eficaz, baseado na e**X**treme **P**rogramming.

Desenvolvimento

“A conclusão é simples: se um projeto de 200 pessoas tem 25 gerentes que são os mais competentes e experientes programadores, demita a tropa de 175 e coloque os gerentes de volta à programação”.[2]

1 "As a result of some conflicts regarding current and future development direction, xHarbour will offer an alternative for those interested in a more aggressive & less conservative approach". Ron Pinkas

2 "The conclusion is simple: if a 200-man project has 25 managers who are the most competent and experienced programmers, fire the 175 troops and put the managers back to programming".

Frederick P. Brooks, Jr.
“O desenvolvimento de software foi feito para ser divertido. Se não é, o processo está errado”.[1] Pete McBreen

Para iniciar o processo, executamos uma compilação do APLICATIVO sem nenhuma alteração. A primeira compilação dos fontes no novo compilador serviu para identificar os pontos mais críticos, uma vez que o compilador interrompeu o processo a cada erro, dando-nos uma idéia de onde estariam as nossas necessidades de correção. Identificamos que nosso maior problema era o fato de utilizarmos muitas bibliotecas de terceiros, devido às limitações da biblioteca padrão do Clipper.

Quando listamos as funções de terceiros que impediam a compilação, procedemos a uma pesquisa para encontrar substitutos. Muitas funções existiam no xHarbour, mas ou tinham outro nome ou tinham outra assinatura de parâmetros. Para manter o histórico de correções sem ser necessário alterar cada fonte, criamos um fonte geral onde escrevemos a assinatura e retorno de todas as funções de terceiros que impediam a compilação com o seguinte padrão:

```
FUNCTION <nome da função>()
RETURN .T.
```

Utilizamos o retorno .T. pois a função poderia estar numa estrutura de decisão. Desta forma, pudemos continuar a compilação até o final e focar apenas em encontrar os substitutos destas funções.

Para mantermos o uso de comandos de usuário como em Clipper, optamos por criar um arquivo de cabeçalho geral e o incluímos em todos os fontes. Isto nos poupou tempo precioso, já que não tínhamos uma documentação apurada do APLICATIVO. No entanto, este arquivo aumentou substanciamente o tamanho do executável final, mas esta era nossa última preocupação ("Faça-o funcionar, faça-o corretamente, faça-o funcionar mais rápido e então, faça-o o menor possível".[2]Kent Beck).

Comandos como mensagens de alerta, títulos e rodapés de tela estavam descritos em cabeçalhos. Assim, retiramos do cabeçalho padrão os comandos comuns e mantivemos apenas as traduções do APLICATIVO. Posteriormente, esta decisão tornou mais fácil a solução dos problemas.

A compilação após a criação do fonte de conversão e da inclusão do cabeçalho geral foi um sucesso. Com estas alterações simples, pudemos verificar o tamanho estimado do executável final e simular uma execução da primeira e segunda telas do APLICATIVO. Isto deu ao time uma sensação de vitória e motivou-o a seguir em frente.

Testes Unitários

"Controle de qualidade envolve o monitoramento de resultados específicos do projetopara determinar se eles concordam com a qualidade padrão relevante e a identificação de métodos para eliminar as causas dos resultados insatisfatórios. Ele deve ser executado durante todo o projeto. Resultados de projetos incluem tanto os resultados de produto, como as entregas previstas, quanto os resultados da gerência do projeto, como custo e prazo".[3] Project Management Institute - PMI

Dividimos o trabalho restante em partes, principalmente para manter a motivação alta. A primeira parte consistia em executar rotinas de acesso a bases de dados e verificar os erros. Algumas das funções de terceiros eram usadas nestes pontos

1 "Software development is meant to be fun. If it isn't, the process is wrong".

2 "Make it work, make it right, make it fast, make it small".

3 “Quality control involves monitoring specific project results to determine if they comply with relevant quality standards, and identifying ways to eliminate causes of unsatisfactory results. It should be performed throughout the project. Project results include both product results, such as deliverables, and project management results, such as cost and schedule performance”.

e tivemos que escrever o corpo das funções no fonte de conversão. Como ainda não tínhamos usado o xHarbour com índices CDX, tivemos que testar e corrigir as chamadas de abertura(USE), importação (APPEND), criação de índices (INDEX ON) e fechamento (CLOSE) de arquivos, além de verificar as funções de queda de energia e reindexação.

Em seguida, ao executar os relatórios, deparamo-nos com problemas de acesso às portas seriais, pois não utilizávamos drivers de impressoras. Como o APLICATIVO tornara-se 32-bit, não necessitava do programa de impressão, criado exclusivamente para este fim, que tínhamos no Clipper. Este programa era necessário porque o Windows não entendia quando o Clipper liberava a porta serial. Quando identificávamos que o sistema operacional era Windows 2000 ou superior, os relatórios eram enviados para um arquivo texto e, então, o APLICATIVO chamava o programa, que lia o arquivo texto e o enviava para a impressora. Este problema não ocorria até o Windows 98 porque o controle de portas não era feito pelo sistema operacional, assim como nos MS-DOS.

Com o xHarbour, alteramos os fontes dos relatórios para eliminar a chamada ao programa de impressão e enviar diretamente para as portas seriais. O novo problema foi a diferença entre a velocidade de envio e a de impressão. Parecia que o buffer da impressora estourava, causando uma paralização na impressão de relatórios extensos.

Solucionamos o problema no cabeçalho padrão. Já que sempre utilizávamos o comando `@ ... SAY` nos relatórios, convertemo-o para:

```
#command @ PROW()+<n>, <col> SAY <xpr> =>;
          Set( 20,"SCREEN" );
         ;Inkey( 0.2 );
         ;Set( 20,"PRINTER" );
         ;DevPos( PROW()+<n>, <col> );
         ;DevOut( <xpr> ).
```

Isto provocou um intervalo de 2 décimos de segundo no envio de cada linha, o suficiente para a impressora liberar o buffer sem atrapalhar o desempenho da impressão. É possível que este problema não aconteça em outras impressoras, mas não tivemos outros modelos para teste.

Como usávamos funções de terceiros na interface de arquivos para geração, leitura e compactação, tivemos sérios problemas neste ponto do trabalho. Para resolver o problema de compactação, que usava o formato LZH, gerado por um programa externo, utilizamos o formato ZIP, cujas funções são contribuições de desenvolvedores do xHarbour. Esta mudança significou o aprendizado das novas funções e alteração dos fontes, um a um, quando a rotina requeria compactação.

Para resolver o problema de acesso de baixo nível a portas seriais, conseguimos a biblioteca HBCOMM, adaptada pelo Luiz Rafael Culik Guimarães. Como ela não continha todas as funções utilizadas por nós, tivemos que adaptar o APLICATIVO para evitar o uso destas funções, sem a perda das funcionalidades já intrínsecas.

Quando conseguíamos passar por uma etapa do teste unitário, compilávamos o APLICATIVO e o liberávamos para a equipe de testes com algumas recomendações do tipo "não façam dois suprimentos em seguida pois o APLICATIVO ainda não consegue apagar a interface". Isto sinalizava para a equipe de testes que eles deveriam se preparar para testar todas as funções do APLICATIVO, exceto as que nós sabíamos que dariam errado. Muitas vezes, estas recomendações auxiliavam até a equipe de suporte. Quando fizemos a migração para xHarbour, utilizamos uma versão não-estável do sistema, o que nos causou muita dor de cabeça. Tivemos que, além de migrar tecnologias, avaliar os erros de lógica e de execução do programa anterior, corrigindo por vezes erros "clássicos" (que ocorriam há muitos releases) e que nunca tinham sido resolvidos. Quando entendemos isto, fizemos questão de que o teste realizasse os caminhos mais críticos do APLICATIVO para que esta versão fosse a melhor já produzida, já que tínhamos que rebater o fracasso da versão anterior.

Testes de Integração

"Em primeiro lugar, o que é um teste? Um teste é a aplicação de uma entrada de algum usuário ou sistema num estado particular da aplicação para definir se a resposta da aplicação é a esperada ou é uma falha em algum sentido – isto pode produzir uma resposta ou saída incorreta, nenhuma saída ou a parada completa da aplicação. Um teste deve avaliar a entrada e o contexto que a aplicação deve atender com a saída esperada".[1] Mike Holcombe

O APLICATIVO não trabalha em rede. Sempre tínhamos, portanto, que nos preocupar com a interface entre os dois módulos, enviadas por disquetes. As informações trocadas pelos módulos são cruciais dentro da unidade de atendimento e, em dois momentos específicos, podem impedir o seu funcionamento. A incorporação do disco de fechamento e o suprimento/recolhimento com o disco de movimento são operações diárias e refletem-se diretamente no saldo financeiro da unidade de atendimento.

A equipe de desenvolvimento e teste efetuou várias idas-e-vindas até que estas interfaces atendessem aos requisitos. A falta de conhecimento sobre o negócio das unidades de atendimento foi um grande empecilho para a solução mais rápida destes pontos. Aliado ao fato de que estas interfaces necessitavam de intervenções específicas, pois usavam compactação e acesso a arquivos de baixo nível, estas funções de terceiros que estávamos testando, o desconhecimento das funcionalidades do APLICATIVO gerou um tempo maior de correção. Era necessária uma vigilância intensa sobre os novatos pelos que conheciam o trabalho realizado nas unidades de atendimento. Citando McBreen, “no ofício tradicional, a aprendizagem leva muito tempo. O ofício de construção de software não é diferente. A razão para isto é que o modelo de ensino de “higienização” rápida não funciona bem para um ofício. Aprender um ofício requer muito trabalho braçal supervisionado, com observações feitas pelos dois lados. O aprendiz precisa ver como o mestre executa uma tarefa tanto quanto o mestre necessita verificar o trabalho do aprendiz”.[2]

Após os testes de integração, solicitamos que a equipe de teste simulasse uma unidade de atendimento para imaginarmos como seria o processo desde a instalação até o dia-a-dia de trabalho. Com isto, forçávamos o APLICATIVO em um ambiente comum mas controlado. Se descobríamos uma falha, corrigíamos e recomeçávamos o processo. Houve dias em que fizemos movimentações de 4 ou 5 dias de trabalho para simular rotinas de segurança e de queda de energia, freqüentes nas unidades de atendimento mais isoladas.

Teste Final

"Um indivíduo sem informações não pode assumir responsabilidades; um indivíduo que recebeu informações não pode deixar de assumir responsabilidades". Jan Carlzon
“Nós pousaremos na Lua antes de 1970”.[3] Jonh F. Kennedy

Quando decidimos que já era o momento de ir para a produção, entramos em contato com algumas diretorias e solicitamos o apoio e indicação de uma ou duas unidades de atendimento para a implantação da nova versão. Esta versão saía como

1 “First, what is a test? A test is an application of some user or system input in a particular state of the system to establish if the system response is what is expected or is faulty in some sense - it might produce an incorrect response or output, no output, or the system might crash. A test must comprise both the test input and the context that the system must satisfy together with the expected output”.

2 "In the craft tradition, apprenticeship took a long time. Software craftsmanship is no different. The reason for this is that the quick “sheep dip” schooling model doesn't work well for a craft. Learning a craft entails lots of supervised and observed hands-on work, with the observation going both ways. The apprentice needs to see how the craftsman does a task just as much as a craftsman needs to oversee the work of the apprentice".

3 “We will land on the moon before 1970”.

solução para a anterior, que estava causando muito tumulto e trabalho para o suporte nacional e regional. Todos os coordenadores do projeto, portanto, concordaram com o teste. Sugerimos uma unidade de atendimento com poucos guichês e movimento, já que a experiência anterior não tinha sido muito feliz. Incluímos uma unidade de atendimento próxima ao local de desenvolvimento do APLICATIVO para que pudéssemos nos deslocar facilmente para identificar problemas, e isto foi extremamente necessário nos dois últimos dias do teste piloto.

Durante 2 semanas, monitorávamos as unidades de atendimento diariamente e liberávamos versões também neste ritmo. Todos os dias, as unidades de atendimento recebiam atualizações após o expediente, normalmente às 21h, para que não atrapalhasse o processo de atendimento do dia. Como algumas deixavam as máquinas desligadas, não conseguíamos atualizar as máquinas durante a noite e, tínhamos de fazê-lo logo no início da manhã, antes das 9h, quando a unidade inicia o atendimento ao público. Sempre utilizávamos o mesmo método de atualização usado no início para diminuir os pontos de erro. Se tivéssemos que criar formas diferentes de atualização, poderíamos criar um problema que só perceberíamos quando fosse tarde demais. Isto nos foi ensinado a duras penas quando fizemos mudanças no método de distribuição da versão anterior.

Depois deste período de teste em produção, com o apoio das agências-piloto e dos coordenadores regionais, iniciamos o processo de distribuição. Desta vez, optamos por atualizar as unidades de atendimento críticas, isto é, aquelas que estavam utilizando a versão anterior. Isto nos daria ânimo para continuar o projeto pois a resposta das agências-piloto, que também utilizavam a versão anterior, havia sido muito positiva.

Conclusão

"O problema-chave com a engenharia de software é que ela nos deixa esquecer sobre as pessoas que desenvolvem software. A promessa implícita da engenharia de software é que, se nós podemos definir um processo quantificado e sistemático, qualquer pessoa pode obter sucesso no desenvolvimento de software. Esta idéia está errada. Como o estudo de campo demonstrou, mesmo com um processo, desenvolvedores excepcionais são vitais para o sucesso dos projetos. Nós temos que focar nossa atenção em como suprir os desenvolvedores de software para que eles, também, possam obter a excelência. Como parte disto, nós devemos questionar o que nós queremos dizer com um processo sistemático de desenvolvimento de software".[2] Pete McBreen

A motivação de ultrapassarmos a fronteira do Clipper nos impeliu ao mundo de 32-bit com o xHarbour. Os projetos do APLICATIVO em Linux, do APLICATIVO em LAN e do APLICATIVO com visual 3D agora são mais viáveis. O método usado para alcançarmos os objetivos mostrou-se eficaz. Conseguimos com que os desenvolvedores ficassem estimulados mesmo com as adversidades de aprender novas tecnologias em prazos curtíssimos.

Por fim, achamos que a experiência de conversão do APLICATIVO com xHarbour com a metodologia baseada em eXtreme Programming nos fez buscar a qualidade no desenvolvimento de software, e nos garantiu cumprir nossas metas de escopo, prazo e custo.

Bibliografia

1. ADAMS, Douglas. **O guia do mochileiro das galáxias**. Tradução de Carlos Irineu

2 “The key problem with software engineering is that it lets us forget about the people doing the software development. The implicit promise of software engineering is that if we can just define a systematic, quantified process, anyone can be successful at software development. This idea is wrong. As the field study showed, even with a process, exceptional developers are vital to the success of projects. We need to focus our attention on how we nurture software developers so that they, too, can excel. As part of doing that, we need to question what we mean by a systematic software development process”.

da Costa e Paulo Henrique Britto. Rio de Janeiro: Sextante, 2004. Título original: The hitchhiker's guide to the galaxy.
2. BECK, Kent. **Extreme Programming Explained: Embrace Change**. [S.l.]: Addison-Wesley, 1999.
3. BROOKS, Frederick P. , Jr. **The mythical man-month: essays on software engineering – Anniversary ed**. [S.l.]: Addison-Wesley, 1995.
4. CARLZON, Jan e Langerström, Tomas. **A Hora da Verdade**. Tradução de Maria Luiza Newlands da Silveira. Rio de Janeiro: Sextante, 2005. Título Original: Moments of Truth.
5. HOLCOMBE, Mike. **Extreme Programming for Real: a disciplined, agile approach to software engineering**. [S.l.]: Prentice Hall, 2003.
6. MCBREEN, Pete. **Software craftsmanship: the new imperative**. New Jersey: Addison-Wesley, 2002.
7. PROJECT MANAGEMENT INSTITUTE. **PMBOK®**. [S.l.: s.n.], 2000.

Referências

xHarbour	http://www.xharbour.org
Harbour	http://www.harbour-project.org
xHarbour.com	http://www.xharbour.com http://www.xharbour.com.br

*O autor é analista de sistemas deste aplicativo desde 1998. É, também, o coordenador da equipe de desenvolvimento responsável pela conversão. É defensor do software livre e utiliza muitos aplicativos opensource no dia-a-dia. Ele também é responsável por todas as traduções livres deste ensaio.