Python桌面应用程序开发的心得体会

引言

随着互联网技术的发展和智能终端设备的普及，软件应用逐渐从桌面程序向移动网络化迁移。虽然如此，一些工程应用的软件还是倾向于独立的桌面应用程序的形式。在这里，我将曾经曾经桌面开发的一些心得整理出来，与大家分享。希望能对专业人员有所借鉴，从中找到不一样的开发思路；对于初学者有所帮助，节约一些时间和精力。为什么这么说呢？我的专业是化学工程的一个分支——过程系统工程，在研究生阶段做项目开发过一些软件，走过一些弯路，也有一些收获。在学校最先学习的是用Visual Studio开发C程序，后来做流程仿真的项目，自学了C++，摸索使用MFC类库开发界面。这个项目开发得很辛苦，其中很多时候都在熬夜修改。一方面由于开发经验不足，另外一方面由于需求不明确，但是当时年轻精力充沛，印象中开发软件就是不停地改啊改。最终虽然完成了项目，但是对于开发过程很不满意。俗话说“穷则思变，变则通”。后来学习了Python语言，并摸索着用这个语言来做项目开发软件，有了之前的经验教训，后来的项目开发顺利了很多。总得来说，语言和工具的优势对于软件开发，尤其是人员很少的项目来说，还是很有帮助的。因此，将使用Python语言开发桌面应用过程中逐渐摸索得到的心得体会总结在这里与大家分享。

成长历程

从学校的教育之下，最先学会的就是使用微软的开发工具。

开发环境是Visual studio，主要类库是 MFC。

后来自学Python语言，并用它用来开发桌面应用。

开发环境为是Eclipse + PyDev

主要用到的模块及工具组合为Python + SciPy + NumPy + wxPython + SWIG + MinGW + py2exe + I18N + NSIS + Subversion

文档方面 Epydoc + reStructureText + Sphinx

测试方面 doctest

下面进行详细说明：

用SciPy，NumPy进行数值计算方面的开发。
用wxPython进行界面方面的开发。
用SWIG包装已有的C代码作为动态链接库。
用MinGW提供的工具链来完成C代码的编译。
用py2exe将Python代码打包成独立的可执行二进制程序。
用I18N进行翻译满足不同的界面字符串变更需求。
用NSIS自动将程序打包为安装程序。
用Subversion进行版本控制，随时随地进行代码修改或者回滚到之前的功能。

用Epydoc生成模块的文档，用reST标记语言来写帮助文档，用Sphinx来组织文档。测试方面使用doctest模块。我们都知道文档和测试方面的工作非常重要，但是在实际项目开发中这些方面的投入会比代码开发少一些。在开发人员有限，有的时候可能就只有1名研究生在做开发，此时遇到导师和用户不断催促着要拿出软件的时候，这方面的时间精力的投入就更没法保障。

需要指出，这里选择的模块都有替代。比如wxPython是一个优秀的跨平台的GUI框架，是wxWidget的Python绑定。GUI框架很多，其中同样优秀的还有Qt的Python绑定，pyQt以及pySide。wxWidget和Qt都是优秀的开发库，选择wxWidegt，还是Qt更多的出自个人爱好。比如我就不喜欢程序代码中有太多的大写字母，而Qt的类库中使用的大写字母比wxWidget多，因此我选择wxPython。虽然如此，但是客观的说，Qt的开发文档更丰富。
同样版本控制软件也有很多，比如CVS，Mercurial，Git等等。
二进制打包程序有很多，比如与py2exe类似的PyInstaller，cx_Freeze等，也可以使用Pyrex，shedskin先将Python代码翻译为C/C++代码，然后再编译为可执行的二进制文件。

看到这里你应该已经能推测到本文覆盖的内容了，如果你还是比较感兴趣，那么欢迎继续阅读，如果涉及的内容，你都比较熟悉，那么大可不必再在这篇文章上花费时间啦。

文本化

适当的开发工具可以极大的提高开发效率，同样重要的还有开发过程中的思想理念。前面介绍了语言开发涉及到的模块和工具，这里写一些对于桌面应用程序开发过程的思考。

简单来说就是：文本化、模块化、敏捷化、多元化、版本化、国际化、自动化。

首先是文本化，“一切皆文本”，这句浅显而又内涵丰富的话，是在学习程序开发很长一段时间之后，才体会到它的深刻含义的。图形界面的开发不单单是用鼠标拖拽，图形界面也是可以用文本来描述的。配置文件和工程文件都可以用文本来描述。程序代码在本质上都是文本文件，任何带有文本编辑功能的软件都可以用来书写代码。一个熟练使用文本编辑器对于开发人员来说非常重要，如果你还没有找到一款趁手的编辑器，那么不妨尝试一下Vim，当然也有人会推荐Emacs。沿着选择编辑器再往下写就跑远了，我们还是来看看配置文件和工程文件的构造。

配置文件

配置文件是典型的文本文件。使用配置文件，程序更加灵活。比如，开发自动更新模块时，将新版软件的下载地址放在配置文件中。这样的自动更新模块，可以几乎不加修改用在其他不同的程序中，在使用时，只需要修改程序的配置文件即可。配置文件的格式有很多，我倾向于使用ini格式的配置文件，相比xml简单易读，而且配置文件的解析和读写可以采用Python自带的configparser模块。看到有的程序，如Django，使用Python文件作为配置文件，或许也是一个可以选择的方案。

工程文件

工程文件也可以用文本来描述。工程文件的描述可以使用类似于配置文件的ini文件来描述。导入导出的工程，就是那个工程文件描述文件加上一堆资源文件，为了使用方便，导出的时候自动压缩为一个zip文件，导入的时候自动解压zip文件，我们又得到文本描述文件。压缩解压过程可以使用Python自带的zipfile模块。

模块化

模块化有两方面的意思，一个是在程序开发过程中不同功能实现之间保持模块化，将代码的耦合程度讲到最低，方便开发和维护；另一方面是指，尽量使用已有的成熟的模块，“不重复制造轮子”，“站在巨人的肩膀上”。花一些时间研究研究使用LGPL协议，BSD协议，Apache协议，eclipse协议的优秀软件包，时常逛逛sourceforge.net，多了解一些实用的软件包，肯定能为软件开发过程带来便利。

图形界面也可以实现模块化，每一个控件都做成一个单独的模块，修改方便，组合也方便。

虽然MFC的设计饱受诟病，但是视图和文档结构对于软件设计还是很有帮助的，这样的设计将界面显示与程序计算隔离开分别设计。

软件界面的布局使用程序代码来描述，不推荐完全依赖使用拖拽生成的界面。如果对界将面要设计得更加灵活，可以使用中间层，用特定的标记性语言来描述界面的外观。使用中间层描述界面，这项工作更像是在开发出第一版软件之后做的事情，因为，只有这个时候基本确定软件界面是什么样子的，有什么地方需要改进。

敏捷化

这里的敏捷化是指开发效率至上，“先完成，再完善”。现在可以说，开发效率远比运行效率重要。因为这是一个飞速发展的时代，已经不是大鱼吃小鱼，而是快鱼吃慢鱼的时代。做项目实现想法，原型很重要。很多时候导师或者用户判断一个想法是否可行，不是看怎么说，而是看看是不是能做出来，做出来之后是不是能满足需求。

计算机硬件在不断改进，根据摩尔定律，计算能力每18个月就翻一番。当运行速度不够的时候，首先应该增加硬件方面的投入，对于企业和大的研发部门，增加一台计算机的成本很多时候不高于开发人员工作一个月的成本，而且，增加一台计算机会产生实实在在的计算能力的提升，而开发人员再开发一个月也不一定能够显著的提升性能，有的时候只会再耽误一个月的进度。

另一方面，如果确实需要提升运行方面的性能，那么最有效的方法是算法方面的改进，如果使用的是成熟的算法库，那么接下来需要做的就是查资料了解各类算法的适用范围，更换更适用的成熟的算法，或者对工作

进行细化，预先判断问题可能存在不同类型，做一些预处理工作，针对不同的问题类型使用特定的成熟算法。当然，在这个改进过程中，可以让用户或者客户先使用之前的版本，也算是一个测试过程。软件开发过程是一个持续不断的改进过程。

多元化

多元化是指使用多种编程语言来完成软件开发，具体来说就是需要多种语言混合编程，充分发挥各自的特长。使用多种语言编程具有两方面的优势：充分利用现有代码；提高软件运行效率。

利用现有代码

如何利用现有代码，对于在研究所或者小实验室的非专业程序开发人员来说，总是一个令人头疼的问题。前面的开发人员水平各异，开发过程中又为了更快地完成任务，很多代码能够运行，但是阅读起来比较困难。因此，如果之前的代码可用，那么就使用这些代码，将这些代码包装成模块，或者作为动态链接库来使用。一方面，可用充分利用前人的工作基础，另一方面，如果以后需要改进，那么在项目基本完成可用的时候，再找时间来改写、重写这些代码也是来得及的。

提高运行效率

还有一种情况就是，对于当前的运行程序的效率不满意。那么通过分析程序运行的瓶颈所在，然后使用更高效率的语言，比如C语言进行改写瓶颈部分的代码，从而实现提高程序运行效率的目的。这也是在项目基本完成之后，进行完善的过程。正如前辈们所说的“过早的优化是万恶之源”。

脚本语言的误解

很早之前，我对脚本语言有误解，认为脚本语言的效率非常低。这个观念直到自己了解并学会用Python绑定C代码的时候才有了改变。

做工程计算的人大多喜欢用MATLAB，因为一个简单的命令就能完成复杂的数学运算。在刚开始写程序的时候，什么都是自己做，主要原因是知识面太窄，因为关于数学理论学习的很深入很透彻，但是不知道有实际的完成相应任务的库。例如前面说道的科学计算库，使用Python使可以使用SciPy科学计算库。

之前也对MATLAB、Python脚本语言存在偏见，认为速度可能太慢。直到后来随着知识的增长，才知道当时认识的肤浅。MATLAB计算的底层库很多是C、Fortran之类的语言编写的，包含了例如BLAS，LAPACK，ODEPACK，UMFPACK等等的软件包。MATLAB语言对这些科学计算库进行了封装，计算的时候调用的就是这些库，效率很高，而且底层这些库都在不断的发展和改进，MATLAB也不断将其吸收消化。同样，SciPy和NumPy库也是对这些知名算法库的包装。因此，使用这些模块，不但效率高，而且很放心，不用担心其中的bug。

版本化

版本控制是软件开发的必备工具，能够清晰有序的保管代码，不用担心代码的丢失，在需要变动的时候也可以快速恢复到之前的工作点。

在软件开发过程中，很多时候需求不明确，摸着石头过河。有时候，导师或者企业领导一句话，这个功能要改会之前的样子，或者要删除，此时，从版本控制软件中就能很方便的找到需要修改的地方。

国际化

国际化是使用I18N技术，来显示界面上的字符串。国际化，使用po，mo文件来更新界面显示的内容，这样做有两方面的考虑：一个是国际化，如果软件推向国际，那么英文版是必须的，这样是很好的解决方法。另一方面，在实际中经常遇到，在向导师和企业领导展示软件的时候，他们有时候对于软件界面控件上的名称不满意，需要修改成特定的称呼，或者专用的词汇。使用I18N技术，这个时候就可以很容易的修改，在.po文件中找到对于的字符串翻译成指定的内容，不用从代码中找到每一处进行修改。这样修改可以保持修改内容的一致性，也可能会地方有遗漏忘了修改。

自动化

自动化是一个很空的命题，包含的内容很丰富。我的理解就是尽量使用脚本，尽量使计算机按照指示进行批处理操作。安装程序可以使用脚本自动生成，测试用例可以自动进行测试，程序的说明文档用户的帮助文档也可以使用工具自动生成。

安装程序自动化

将制作安装程序的过程写成脚本，那么每次修改代码之后，只需要执行一次脚本就可以制作好可以分发的安装程序，尤其在情况紧急时，能帮上大忙。NSIS安装脚本可以方便的制作安装程序。

测试用例自动化

测试用例自动化是指在程序开发过程中，有的地方就用doctest写好测试用例，一方面可以在后续的改动中确保修改的内容，不会对之前修改造成影响；另一方面，其他人在调用其中的函数时可以清晰看到具体的调用方法。而且自动测试用例也会作为文档的一部分出现，使得文档内容更加丰富。

文档生成自动化

程序说明文档和使用帮助文档很重要，写文档是一件枯燥的事情，程序说明文档可以在写程序的时候就着手来写。Epydoc是一个Python模块文档自动生成工具。它会根据用户编写的Python模块中的文档字符串自动生成这个模块的文档。

用Sphinx组织用reST标记语言书写的帮助文档，这种标记性的语言功能丰富，书写方便，而且可以轻松生成latex、html文件。

后记

Excel + VB 是不是一个好选择？从程序开发的角度来说似乎不是很完美，从实用性的角度来说，这是可以考虑的一种方案。MATLAB自带的GUIDE进行图形化界面编程也很方便，用TraitsUI，Chaos也可以快速开发图形化的应用程序，这样的组合还有很多，根据具体情况来选择最合适的搭配。

以上只是一段时期用Python开发桌面应用程序的一些心得体会，可能更适用于非电类专业的研究生在短时间内为研究项目开发桌面应用软件，其中提到的工具和开发理念也不一定是正确的和普适的，因此，仅供参考。虽然如此，但是有一点是确定无误的——随着时间的发展，还会不断涌现出更多更好的开发工具，也会发展出更多更实用的开发理念并广泛传播。

写程序是工作的一部分，而工作又是生活的一部分。

王杭州2013年5月于清华园