A SOFTWARE ENGINEERS' PERSPECTIVE TO APPLICATION DEVELOPMENT IN MATLAB

Sergei Chevtsov, Michael Zelazny, Greg White, SLAC, Menlo Park, USA

In this paper, we describe reasons for, and consequences of, the decision by the LCLS software development team to create physics applications in Matlab. The following discussion is neither objective nor complete, but only offers an overview of possible obstacles that can be encountered during development of Matlab software. We hope that due to similarity among projects at national laboratories around the world, our experience at SLAC will be useful for fellow application developers.

INTRODUCTION

In March 2006, our team at SLAC agreed to use Matlab to develop a high-level applications suite for commissioning of the LCLS linear accelerator, which was scheduled for March 2007. In the course of one year, we had encountered and solved many problems with Matlab that we would like to illustrate in this paper. As a case study, we use the application for acquiring, processing, and managing electron beam images (ImgMan).

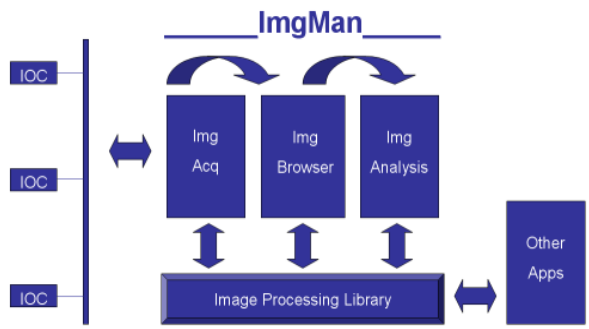

Figure 1: ImgMan

ImgMan consists of an image processing library and three autonomous components with graphical user interfaces (GUIs). The first component features a GUI for processing live images and acquiring image datasets from EPICS IOCs. The second component is an image browser for managing local datasets of images. The third component features a GUI for analyzing retrieved images. The underlying image processing library is used by other physics applications as well [1].

MAKING THE DECISION

In March 2006, we had Microsoft Visio mock-ups of how ImgMan was going to look like to the end user. However, we knew very little about how the communication with IOCs would work, what data our physicists wanted to extract from images, and how to organize our code for reuse. We concentrated on the following criteria for our decision to develop ImgMan in Matlab.

Rapid development

After talking to Matlab developers who were mostly writing scripts for their own use, we expected to implement the functional requirements in Matlab rather quickly. However, we had no idea how difficult it would be to realize the non-functional requirements of a quality software application, such as: robustness, reusability, comprehensive error reporting, user preferences, and online help.

ChannelAccess

Our colleagues at SLAC are authors and maintainers of labCA, a stable and very simple Matlab client library for ChannelAccess. We estimated that we could easily integrate it into ImgMan.

Working with physicists

Overall, the most vocal impulses for using Matlab for ImgMan came from physicists, because they wanted to write their own image processing algorithms. We also expected that the collaboration with physicists would help us to create user-friendly screens with intuitive labeling.

IDE

Matlab's integrated development environment (IDE) was expected to accelerate implementation of ImgMan, because it provided a code formatter, a debugger, and a built-in console for running Matlab scripts.

Workshop

In March 2006, we attended a Matlab workshop and learned about the 24-hour support, the wide deployment of Matlab scripts in the industry, as well as exhaustive online help and documentation. We were ready.

CHALLENGES

The issues below were encountered at different times during our development cycle and are ordered by roughly how much time was needed to either get accustomed to, or to work around them. The first issue had the biggest impact on our project.

Dynamic typing

In Matlab, a variable can be instantiated without explicitly specifying its type. When we dealt with scalar values, this led to hardly any problems; some of us even found that this capability helped during prototyping. However, ImgMan's design demanded for image-related data to be stored in complex structures. When a complex

structure was improperly used in a script, Matlab could only report errors (such as a wrong field order or mistyped field names) at runtime. Auto-completion, which can be found in all modern software toolkits, was not available. Instead, we manually had to look up the exact definition of each structure. Only after several months were ImgMan structures so engraved in our minds that this activity became less burdensome.

Refactoring

With constant changes in system and user requirements, many modern software engineering techniques highlight re-factoring as an essential piece of the development cycle [2]. LCLS project was no different as we did not have all requirements set in stone before the commissioning. Unfortunately, the lack of proper re-factoring tools in Matlab made every change in ImgMan, especially during the last few months of development, an excruciating task.

No need to declare variables

Matlab doesn't force programmers to declare variables before using them. Thus, typos could only be discovered at runtime. Eventually, we learned to type out variables only during initialization, and to copy and paste them subsequently.

Working from home

We originally thought that we could develop ImgMan from home by running the IDE on a machine at SLAC and displaying it locally via forwarding X11 over SSH. Then we discovered that Matlab's IDE was based on Java 1.4 Swing framework that has documented performance problems in remote display environment [3]. Basically, the Matlab IDE was completely unusable in our scenario. Even though we found some workarounds, developing ImgMan from home had never become a pleasant experience.

Confusing syntax

Despite the fact that it is learned by many non-professional programmers, we found that Matlab syntax was harder to grasp than expected. The biggest confusion came from correctly using arrays. Matlab features the so-called column and row arrays, whose elements are addressed in different ways. Generally, we overcame this obstacle quickly, but there had been isolated instances of mistakes until the end of the project.

LIGHT AT THE END OF THE TUNNEL

We also had some positive experiences with developing Matlab programs.

Since Matlab is an interpreted language, ImgMan code could be changed without restarting the application. This feature was very helpful during commissioning in front of the physicists.

Most notably, physicists were able to not only provide us with their complex functions for image processing, but also to debug ImgMan on their own. We can proudly say that, at least by the end of the project, our work was conducted in close cooperation and led to close ties with LCLS physicists

CONCLUSION

Due to the lack of many features available in established programming languages, we recommend that only small software projects are developed in Matlab. For more complex tasks, we suggest sticking with traditional, more powerful languages. If a close integration with physicists is desired, support for Matlab scripts could be added to the developed software. Last, but not least, we think that for a successful Matlab project, programmers should be supplied with standalone licenses, so that they can work from home.

REFERENCES

[1] Michael Zelazny, "Electron Bunch Length Measurement for LCLS at SLAC", ICALEPCS'07, Knoxville, October 2007
[2] http://c2.com/cgi/wiki?WhatIsRefactoring.
[3] http://java.sun.com/products/java-media/2D/perf_graphics.html#70248.