# Unified framework for development, deployment and testing of image analysis algorithms

*Release 0.00*

Alark Joshi[1], Dustin Scheinost[1], Hirohito Okuda[4], Isabella Murphy[1], Lawrence H. Staib[1,2,3] and Xenophon Papademetris[1,2]

July 10, 2009

[1]Department of Diagnostic Radiology, Yale University
[2]Department of Biomedical Engineering, Yale University
[3]Department of Electrical Engineering, Yale University
[4]GE Healthcare, Japan

**Abstract**

Developing both graphical and command-line user interfaces for image analysis algorithms requires considerable effort. Generally developers provide limited to very rudimentary user interface controls. These image analysis algorithms can only meet their potential if they can be used easily and frequently by their intended users. Deployment of a large suite of such algorithms on multiple platforms requires that the software be stable and appropriately tested.

We present a novel framework that allows for rapid development of image analysis algorithms along with graphical user interface controls. Additionally, our framework allows for simplified nightly testing of the algorithms to ensure stability and cross platform interoperability. It allows for development of complex algorithms by creating a custom pipeline where the output of an algorithm can serve as an input for another algorithm. All of the functionality is encapsulation into the object requiring no separate source code for user interfaces, testing or deployment. This makes our framework ideal for developing novel, stable and easy-to-use algorithms for computer assisted interventions (CAI). The framework has been deployed at the Magnetic Resonance Research Center at Yale University and has been released for public use.

Latest version available at the Insight Journal [ http://hdl.handle.net/10380/3078]


## Contents

## 1 Introduction

Image analysis algorithms are typically developed to address a particular problem within a specific domain (functional MRI, cardiac, image-guided intervention planning and monitoring, etc.). Many of these algorithms are rapidly prototyped and developed without considerations for a interface (GUI), robust testing, and integration into a large software package. Sometime these features are added later, but require considerable amount of effort on the part of the developer of the original algorithm. This makes it increasingly difficult for deployment and widespread adoption of the newly developed algorithms especially for CAI algorithms where robust testing and easy-to-use interfaces are critical.

BioImage Suite [8] is a comprehensive, multi-platform image analysis suite comprised of many different image analysis algorithms with a focus on epilepsy neurosurgery. In previous versions of BioImage Suite (up to version 2.6 released in November 2008), all algorithms were implemented in C++ and invoked from either command line scripts or GUI modules both written in the Tcl scripting language. However, the command line scripts and GUI modules were two separate implementations of essentially the same algorithm and would invariably diverge without extensive coordination. This required developers to create both command line scripts as well as complex GUIs. Testing became problematic as two new applications need to be tested for each new algorithm. Finally as new algorithms became more complex, basic components (e.g., image smoothing) were often reimplemented instead of using existing implementations of these components.

To address the issues discussed above, we developed a framework that unifies the algorithm that is being invoked from the command line as well as from the user interface. We have chosen a component-based software approach which has been widely used and researched in the field of software engineering [6]. In our framework, a component performs an operation (smoothing, surface extraction and so on) on the specified input data (images, surfaces, transformations). The main algorithm is developed in C++ while its functionality is encapsulated into an [Incr Tcl] object. This allow the user to instantiate the object in a tcl script and handle the input/output as well as the GUI via inherited methods.

With this framework, the developer can focus on the creation of the algorithm and not worry about software engineering aspects needed for CAI algorithms such as testing, integration, and creating customized workflows. The GUI is automatically generated by the algorithm object when the object is invoked. Testing is handled by the algorithm object by specifying the inputs, expected outputs, and the test flag. Since new algorithms are created in the same framework, it is possible to create data workflows where the output of a simple algorithm can then be used as the input of another algorithm. Thus, developers can reuse existing algorithms saving time and reducing programing complexities.

## 2 Related Work

Software development work in the field of medical image analysis has focused on describing the architecture for a specialized setting. Coronato et al. [1] discussed an open-source architecture for immersive medical imaging that used 3D graphics and virtual reality libraries. Additionally, they also include ubiquitous computing principles for context aware interaction with mobile devices. Shen et al. [4] discuss their system which works with stereoscopic displays and uses projectors to provide an immersive experience in environments such as the CAVE.

In the field of medical image analysis, medium to large imaging software such as Slicer3D [5] have a component based approach to developing software that allows for the easy development of user interfaces for developers. Here each algorithm generates an XML file specifying instructions for creating a GUI, which is then read by the main application at run time to create a GUI. One of the limitations of this approach is that an external application creates the GUI which increases the complexity of the external application and implies that the algorithm implementation cannot function as a stand alone application. Medical Imaging Interaction Toolkit (MITK) [3] is a medical imaging toolkit for image analysis, which has some features similar to those in our framework. However, it is intended to be used as a toolkit and "is not intended as an application framework" [3] that can be used for image analysis by users.

The functionality included in all the abovementioned image analysis software systems and others is very similar. Our contribution is in the ability to provide researchers in image analysis an open-source platform-independent framework that allows them to focus on developing new algorithms. The software engineering aspects of interface design, testing protocols, and code reusability are automatically provided to the researcher assisting deployment and widespread adoption of the newly developed CAI algorithms.

## 3 System Overview

Our unified framework allows for easy development, deployment, and overall packaging of image analysis algorithms. Using this framework, developers can effortless create user interfaces and seamlessly test their algorithm on multiple platforms. Novel algorithms can easily be added and custom workflow pipelines can be constructed where each piece of the pipeline is an algorithm that takes an input and performs an operation. Figure 1 shows a flowchart for an image analysis algorithm. Each new algorithm takes a combination of images, surfaces, transformations, and input parameters and produces a combination of images, surfaces, and transformations as outputs.

BioImage Suite algorithms are packaged into a single set of [Incr Tcl] classes. These classes are characterized by two key methods, `Initialize` and `Execute`. In the `Initialize` method, each algorithm explicitly defines three sets: (i) inputs, which are objects such as images, surfaces, landmarks etc., (ii) parameters, which are single values such as integers, strings, filenames, and (iii) outputs, which are also objects. Figure 2 shows a detailed example of an `Initialize` method. The `Execute` method invokes and passes object to/from the underlying C++ source code.

Based on the definition of the input and output sets, the base abstract classes have functionality (which need not be touched by more concrete implementations) to (i) parse command line arguments if the algorithm class is invoked as an application; (ii) automatically create a GUI using the `CreateGUI` method (this method can be overridden by some algorithms to generate a more customized interface); and (iii) perform testing by parsing a test file. These classes can then be used (i) to invoke the algorithm (using an `Execute` method), (ii) to become a components of other algorithms (e.g. the image smoothing algorithm is invoked by the edge

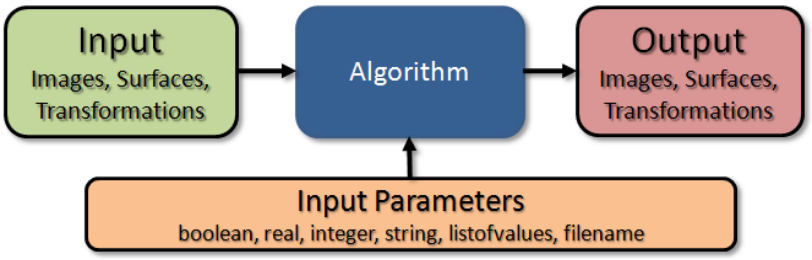


Figure 1: This diagram provides an overview of the new unified framework for image analysis algorithm development. Any image analysis algorithm has a combination of images, surfaces, and transformations (from registrations) that serve as input to the algorithm. The algorithm most probably has some input parameters (which can be specified on the command line or can easily become GUI components). In our framework, the input parameters can be one of boolean, real, integer, string, listofvalues (for drop down options when using a GUI) or a filename. The output too can be of any one of image, surface or transformations.

detection algorithm), (iii) to create a standalone applications with an image viewer and a GUI, and (iv) to integrate individual components into a larger application. Figure 3 shows how the same code is invoked using (i) the command line, (ii) the GUI, (iii) a larger application, and (iv) for nightly testing.

```
itcl::class bis_smoothimage {
    inherit bis_imagetoimagealgorithm
    constructor { } {    $this Initialize  }
    public method Initialize { }
    public method Execute { }
}

itcl::body bis_smoothimage::Initialize { } {
    #name,description,type,object,filename(if applicable),priority (optional)
    set inputs { { input_image "Input Image" pxitclimage  ""    0 } }
    set outputs { { output_image "Output Image"  pxitclimage  "" }}

    #commandswitch,description,shortdescription,optiontype,defaultvalue,valuerange,priority
    set options {
        { blursigma "kernel size [mm/voxel] of FWHM filter size" "Filter Size"  { real triplescale 100 }    2.0 { 0.0 20.0 }  0 }
        { unit      "kernel size unit mm or voxels " "Units" { listofvalues radiobuttons }  mm { mm voxels }   1}
        { radius    "radius factor of the gaussian in voxel units " "Filter Radius" real  1.5   { 0.0 5.0 }  -1 }
        { dimension  "2 or 3 to to do smoothing in 2D or 3D" "Dimensionality"  { listofvalues radiobuttons }    3 { 2  3 }  -999 }
        { testlog   "0: no test log , 1: output test log  " "testlog" boolean   0     { 0 1 }  -1000 }
    }

    #document
    set description "Smoothes an image with a specific gaussian kernel."
    set description2 "Smoothing kernel size blursigma (in mm by default ) represents the FWHM filter size."
    $this InitializeImageToImageAlgorithm
}
#  This checks if executable is called (in this case bis_smoothimage.tcl) if it is execute
# ----------------------------------------------------------------------------
if { [ file rootname $argv0 ] == [ file rootname [ info script ] ] } {
    # this is essentially the main function
    set alg [bis_smoothimage [pxvtable::vnewobj]]
    $alg MainFunction
```

Figure 2: Sample implementation for a image smoothing algorithm. A new algorithm usually requires the implementation of two methods. The first is *Initialize* (shown in detail in this figure) where the inputs, outputs and parameters are defined. The second is *Execute* (not shown) which simply takes the specified inputs and parameters and runs the actual algorithm to generate the desired output. Derived classes can have customized GUI by overriding the CreateGUI method.

## 3.1 Core Classes:

The new framework has at its core the following [Incr Tcl] classes:

1. `bis_option` encapsulates an option value (e.g. smoothness factor, etc.). An option can have a type of: listofvalues, boolean, real, integer, string or filename. Within this class there is functionality for creating an appropriate GUI for each option.

2. `bis_object` encapsulates the input and output objects of the algorithms. The core objects supported are: image, transform (both linear and non-linear), polygonal surface, landmark set and electrode grid.

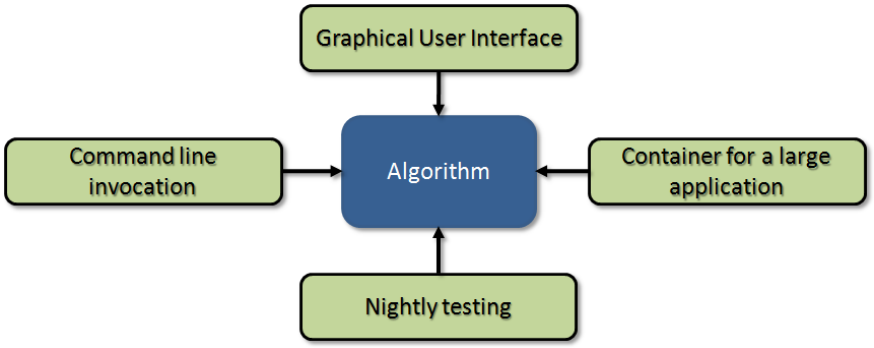


Figure 3: This schematic shows the unified nature of our framework. The command line invocation of an algorithm as well as the GUI use the same algorithm source code and can be controlled using a simple *-dogui* flag. For integration into a larger user application, the same source code is used to give the user a toolkit of similar algorithms that can be used. Additionally, for nightly testing the same algorithm code is invoked with different input parameters that test the algorithm on various platforms.

3. `bis_basealgorithm` is the core algorithm class from which all algorithms are derived. It has all the functionality for manipulating options, inputs and outputs.

4. `bis_algorithm` is derived from `base_algorithm` and adds the functionality needed for taking an algorithm and making it into a component or an executable. More specialized classes are derived from `bis_algorithm` such as `bis_imagetoimagealgorithm` which serves as a base for algorithms which take a single image as an input and produce a single image as an output.

5. `bis_guicontainer` is a derived class of `bis_algorithm` and serves as a parent class for creating multi-algorithm containers (e.g. a tabbed-notebook style GUI where each tab is a separate algorithm).

## 4 Current Status

In this section, we discuss the current status of the new framework. We provide details about the invocation of the algorithms, discuss our nightly testing setup and show an example of a customized data workflow that is being used for processing SPECT data for epilepsy neurosurgery.

### 4.1 Algorithm interfaces

An algorithm can be invoked in three ways: (i) command line, (ii) GUI, and (iii) managed graphical interface. The framework facilitates the invocation of the same code regardless of the manner in which the script is invoked. In Figure 4, we can see an example of a non-linear registration script being invoked in three different ways. Labels A, A1 and A2 show a GUI with different components showing the input parameters. Label B in the figure shows a command line invocation which also provides Unix-style help to the users. Additionally, the same script can be contained in a managed container for a larger application (as shown by label D).

Using this framework the user can use the "Show Command" button embedded in the GUI (shown in Figure 4 label C). The user can familiarize themselves with the algorithm at the GUI level. Then, the user can press this button and get a detailed command line specification for performing *exactly* the same task by invoking *exactly* the same code at the command line. This feature makes it easier for end-users to develop customized batch jobs/pipelines.

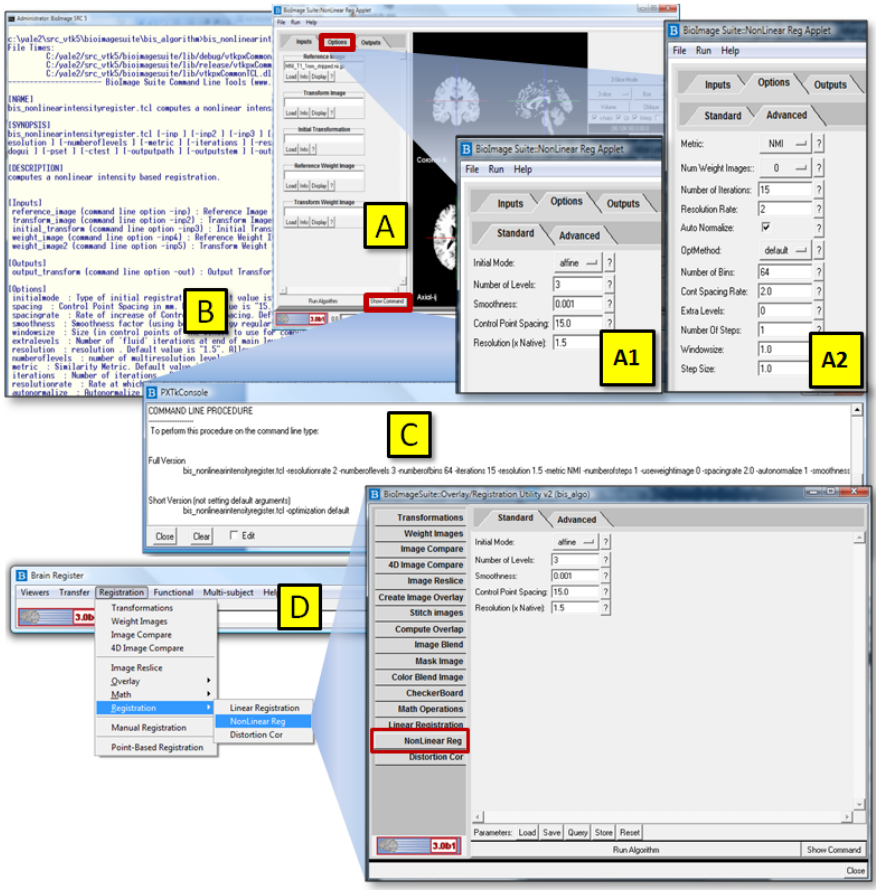


Figure 4: This figure shows all the different ways in which a script can be invoked. Label (A), (A1) and (A2) shows the graphical user interface (A) with the parameters in the standard tab (A1) and the advanced tab (A2). Additionally, the user can click on the "Show Command" button highlighted with a red rectangle that shows how the script can be invoked on the command line (C). The script can also be invoked on the command line (B) and in the situation where incorrect input parameters are provided, a Unix-style help is shown that shows the format for the input and input parameters. Additionally, the script can be contained in a managed framework (D) where it becomes a menu item that invokes the same graphical user interface options as in (A).

## 4.2 Nightly Testing

The implementation of our testing framework allows for easy addition of test cases. For testing, we maintain a list of all the test cases which have a format as follows:

```
algorithm name : input parameters and their values: input files : expected output file
bis_smoothimage: -blursigma 2.0: MNI_T1_1mm.nii.gz : MNI_T1_1mm_sm2.nii.gz
```

When the nightly testing process starts, it goes through and tests each algorithm. For each algorithm, it looks up its name in the first column of the list, and if the name matches then it reads in the remaining arguments and performs the test. As shown above, to test the image smoothing algorithm we specify the name of the script, the input parameters and their values (blursigma=2.0 in this case), the input file name and the expected output file name to compare the output with. The obtained output is compared with the expected output and based on the comparison a "test passed" or "test failed" result is obtained. Therefore, adding more test cases is as simple as adding another line to the list of nightly tests for that algorithm.

Figure 5 shows a composite figure of the nightly dashboard which shows the platforms that the scripts are tested on. As of now, the nightly tests run on Linux (CentOS), Windows (Vista and XP), Power Mac and Mac OS X . On the right is a screenshot of a list of some of the scripts as can be seen on the dashboard. This allows us to readily know whenever a script fails on a particular platform.

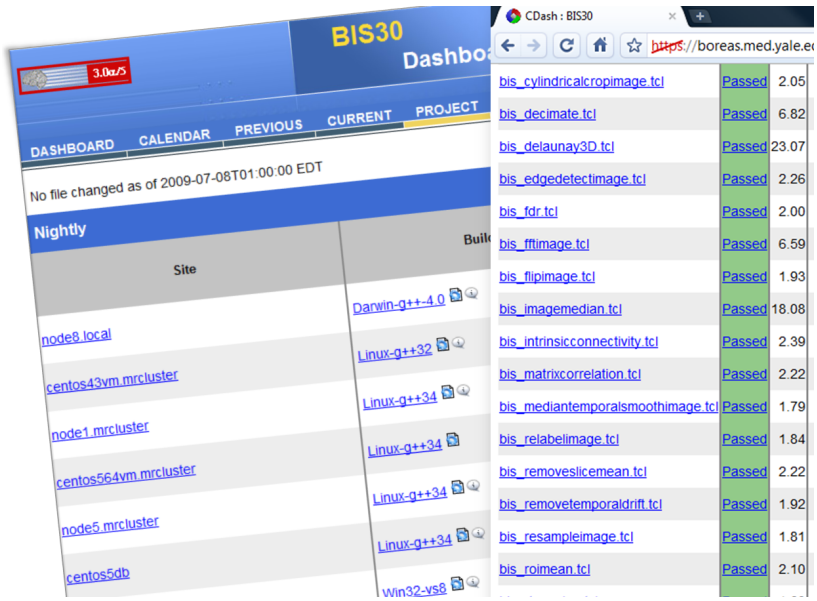


Figure 5: In this figure, we can see the various platforms that the scripts are tested on. This happens on a nightly basis and allows for multi-platform testing. The right image shows a sample list of scripts being tested and their status on a particular platform.

### 4.3 Customized workflow - Diff-SPECT processing for epilepsy

Using this framework, customized workflows can be created to enable the development of complex and streamlined algorithms. In these customized workflows, the output of one algorithm can be used as the input to another algorithm. Here we present an example of a customized workflow for ISASHN algorithm [2] used to assist image-guided surgery research. First, two SPECT images are linearly registered to each other and then nonlinearly registered into MNI space. The registered images are then masked, smoothed, and intensity normalized. A t-test is preformed comparing these images to a healthy normal population. The resultant tmap is thresholded and clustered to produce the final output. This workflow can be implemented as a single algorithm object with it own GUI and testing protocol that sequentially calls other algorithm objects as presented in Figure 6. The algorithm object can be instantiated from our BioImage Suite VVLink gadget to connect to the BrainLAB Vector Vision Cranial system for integration into neurosurgical research [7].

## 5 Conclusion

Using our novel framework, we have developed unified tools for our users that allows for easy to use interfaces and robustly tested algorithms. Additionally, customized workflow pipelines have been created by developers to allow for creation of complex algorithms. With this framework, we envision a more widespread adoption amongst our research group for rapid development of easy-to-use image analysis algorithms and look forward to other CAI contributions to BioImage Suite.

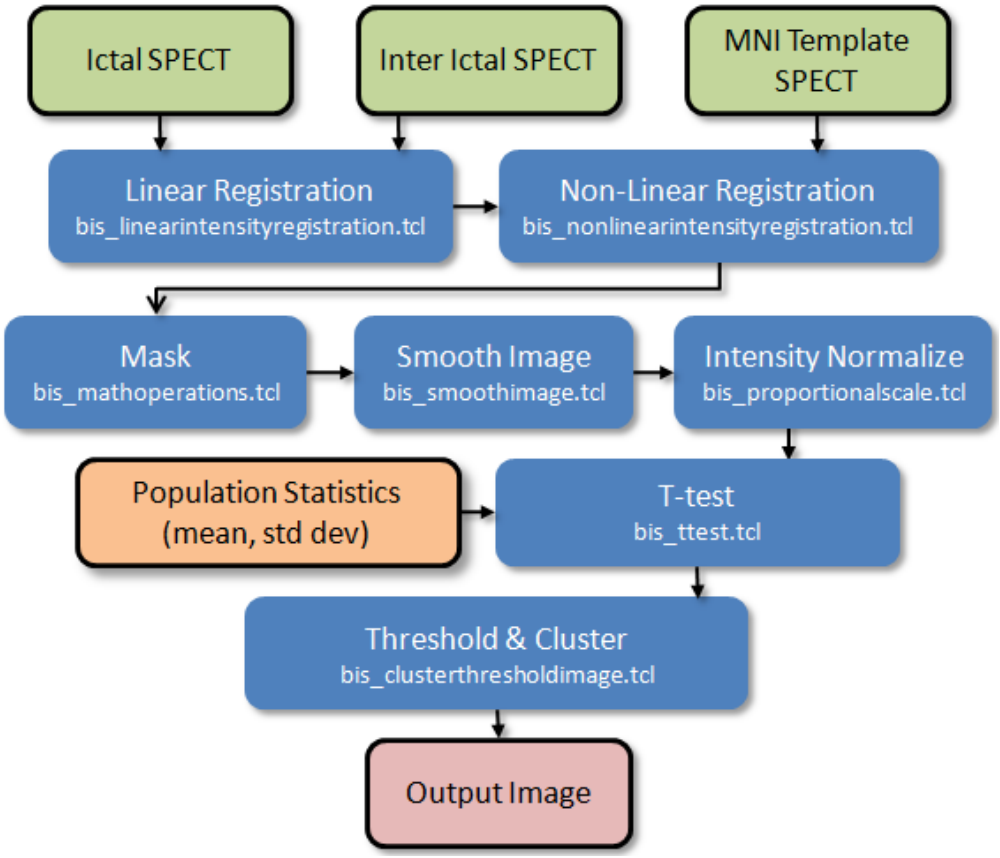


Figure 6: Customized workflow using the unified BioImage Suite framework. Here the algorithm modules are depicted in blue (with the actual script name below it). In this workflow, the interictal and ictal SPECT are first linearly registered and output is then non-linearly registered with the MNI Template SPECT. The result of the registration is then processed using various algorithms (mask, smooth and intensity normalized). Then a t-test is performed with the mean and standard deviation from a control population. The output tmap is then thresholded and clustered to get the final output image.

## References

[1] Coronato A, De Pietro G, and Marra I. An open-source software architecture for immersive medical imaging. In *In Proceedings of the IEEE International Conference on Virtual Environments, HCI and Measurement Systems*, 2006. 2

[2] Scheinost D, Blumenfeld H, and Papademetris X. An improved unbiased method for diffspect quantification in epilepsy. *IEEE International Symposium on Biomedical Imaging ISBI 2009*, June 2009. 4.3

[3] Wolf I, Vetter M, Wegner I, Bottger T, Nolden M, Schobinger M, Hastenteufel M, Kunert T, and Meinzer HP. The medical imaging interaction toolkit. In *Medical Image Analysis*, pages 594–604, Dec 2005. 2

[4] Shen R, Boulanger P, and Noga M. Medvis: A real-time immersive visualization environment for the exploration of medical volumetric data. In *Proceedings of the Fifth International Conference on BioMedical Visualization*, pages 63–68, 2008. 2

[5] Pieper S, Halle M, and Kikinis R. 3D slicer. *IEEE International Symposium on Biomedical Imaging ISBI 2004*, 2004. 2

[6] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, 2nd edition, 2002. 1

[7] Papademetris X, DeLorenzo C, Flossmann S, Neff M, Vives K, Spencer D, Staib L, and Duncan J. From medical image computing to computer-aided intervention: development of a research interface for image-guided navigation. In *Int J Med Robot*, volume 5, pages 147–157, 2009. 4.3

[8] Papademetris X, Jackowski M, Rajeevan N, DiStasio M, Okuda H, and Constable RT. Bioimage suite: an integrated medical image analysis suite: an update. In *SC/NA-MIC Workshop on Open Science at 9th MICCAI Conference*, 2006. 1