IfA/MHPCC Image Processing Pipeline Pilot Project
Software Requirements Specification (SRS)
### Revision History

<table>
<thead>
<tr>
<th>Version/Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR-01</td>
<td>10 Oct 2003</td>
<td>Original version not under document control</td>
</tr>
<tr>
<td>DR-02</td>
<td>13 Oct 2003</td>
<td>Additional detailed module break-down, Section 10</td>
</tr>
<tr>
<td>DR-03</td>
<td>22 Dec 2003</td>
<td>Re-formatted, re-defined for Phase 2 of the Pilot Project</td>
</tr>
<tr>
<td>DR-04</td>
<td>23 Dec 2003</td>
<td>Re-organized image section, added tests, program I/O</td>
</tr>
<tr>
<td>DR-05</td>
<td>05 Jan 2004</td>
<td>Added suggestions and revisions from MHPCC &amp; IfA</td>
</tr>
<tr>
<td>DR</td>
<td>14 Jan 2004</td>
<td>Renumbering to DR format</td>
</tr>
<tr>
<td>00</td>
<td>27 Jan 2004</td>
<td>Assigned to 00 status</td>
</tr>
<tr>
<td>01</td>
<td>09 Mar 2004</td>
<td>Adjusted 3.4 (OS) and 3.9.3 (Mask source)</td>
</tr>
</tbody>
</table>
1.0  SCOPE

This document details the requirements for the Pan-STARRS Pilot Project Image Processing Pipeline (IPP) software that is a deliverable product equivalent to a Computer Software Configuration Item (CSCI). The Pilot Project is a software development effort jointly conducted by a team from the University of Hawaii Institute for Astronomy (IfA) and a team from the Maui High Performance Computing Center (MHPCC), and is considered as a preliminary step in the development of the final Pan-STARRS IPP. More specifically, this document specifies the required coding languages and standards, platform architecture compatibilities, documentation and version control, modules, input/output data, and input/output formats that must be included in the delivered CSCI as well as required testing processes and procedures. This document does not include requirements for the final Pan-STARRS IPP, nor does it specify a relationship between the Pilot Project IPP and the final IPP in terms of pipeline architecture, SW design, or legacy code.

Note: all exceptions to the requirements in this document must be requested from the IfA/MHPCC Software Configuration Control Board (CCB).

The goal of the Pilot Project is to produce a set of basic image manipulation routines (modules) and to explore options for tying the modules together in an efficient and effective way. The primary analysis pipeline defined by this Pilot Project consists of the basic image reduction and object detection stages for a single image.

2.0  APPLICABLE DOCUMENTS

The PanSTARRS document control naming scheme is PSDC-NNN-MMM-VV, where the VV entry specifies the version number. Where documents are described without the version number, the latest version of that series is implied.

2.1  Government Documents

N/A

2.2  Non-Government Documents

PSDC-410-002, IfA/MHPCC IPP Pilot Project Algorithm Description Document (ADD)

3.0  REQUIREMENTS

3.1  CSCI deliverable

All final source code generated for the Pilot Project is to be delivered via CVS. In addition, binaries are to be compiled for the x86 / Linux-32 architecture and delivered via the G-Forge website.
3.2 Programming languages, standards, and conventions

3.2.1 Source code
All source code is to be in C.

3.2.2 Scripting Language
When appropriate, software written in a scripting language is allowed. Perl is the designated scripting language.

3.2.3 Software interface
A user interface is to be provided with appropriate functionality to allow the linking of low-level modules with a simple script. The public API shall consist of the module functions, as noted in section 3.8. All public APIs are to be compatible with SWIG, and associated SWIG interface files shall be generated.

3.2.4 Code standard
The C code is to comply with ANSI Standard C99. Source code files shall use the UNIX line-break convention (LF only).

3.2.5 Coding style standard
The coding style of delivered code should conform to the Sun/Java standard as modified to reflect C language constructs ([tbd document]), along with the following exceptions:

3.2.5.1 In order to enable certain emacs code navigation tools, all functions should be modified to have the opening brace in the first column of the line following the function definition.

3.2.5.2 In-line comments for parameters to functions are recommended in both the function definition block and in the function declaration in the header files. At least one of these two locations must include the commented function parameters. The following example demonstrates the recommended style and the associated line doxygen-compatible parameter comments:

```c
 type function (
    type var1, ///< the first parameter using C++ style comments
    type var2) /*!< the second parameter using C-style comments */
{
    /* the function body */
}
```

3.2.5.3 Perl code shall follow the Perl Style guidelines provided with Perl.

```perl
 type function (
    type var1, ///< the first parameter using C++ style comments
    type var2) /*!< the second parameter using C-style comments */
;
```
3.2.6 Comment conventions

Commenting of delivered code is to be consistent throughout the deliverable, independent of developer, and compatible with Doxygen parsing (see 3.3.2).

3.2.7 UNIX operating system standard

Because the delivered code is required to run within UNIX operating systems (see 3.4), the delivered code should be in compliance with the language-independent UNIX operating system standard, POSIX, Open Group Base Specifications Issue 6, IEEE Std. 1003.1, 2003.

3.2.8 Version control

Source code version control is to be implemented through CVS.

3.3 Documentation

No proprietary documentation formats are allowed. All delivered documentation will be in one of Adobe PDF, HTML, or ASCII text as appropriate (see below).

3.3.1 Design documentation

A Software Design Description (SDD) document is to be provided by MHPCC that incorporates details from this Software Requirements Specification (SRS; PSDC-410-001) and the Algorithm Definition Description (ADD; PSDC-410-002). Design documents shall be delivered in PDF format.

3.3.2 Code documentation

Doxygen is to be used to generate code documentation directly from in-line comments.

3.3.3 Man pages

Man pages are required to provide explicit and detailed definition of all public API software interfaces.

3.4 Platform architectures and operating systems

Makefiles are to be provided with appropriate flags set so that all code compiles without warnings under “gcc –Wall” (gcc v2.95 and higher) for the following platform architectures and operating systems:

- x86/Linux
- Sparc/Solaris
- PPC/OS—X

This “compilation without warnings” requirement is to be extendable to other identified architectures/operating systems with minimal modifications to Makefiles or configuration scripts. The requirement of compiling without warnings includes the allowance that lex-generated code be
filtered to remove specific, known warning. The Perl code shall be produced to run without warnings using the interpreter version 5.8.0 or 5.8.2 compiled without multithreaded support.

3.5 Images

Image files are to be read from disk given a specified file name, not longer than 1024 characters.

3.5.1 Image definition

An image is defined as an image data file representing the (pixel + overscan) region.

3.5.2 Image sizes

Image (including overscan portions) sizes read from disk are to be determined from the header metadata following the FITS standard; pre-defined image sizes coded into the IPP are not allowed.

3.5.3 Image file formats

Both input and output image data files are to be in FITS format. It is to be possible to write image data in both integer and floating point formats (all possible FITS BITPIX values). Input images may consist of flat 2-D files or of 3-D cubes in which the slices of the cubes represent separate cells of an OTA. The number of slices defines the number of Cells in the OTA; in the case of 2-D image files, the number of Cells shall be assumed to be 1.

3.5.5 Image type conversion

For the purpose of calculation/manipulation, all image data is to be converted to an internal floating point format of at least 32 bits.

3.5.6 Image magic number values

A BADPIX “magic” data value is to be defined for output image pixels.

3.5.6.1 Floating point data

For the case of floating point data, IEEE special values of “NaN” shall be used for BADPIX.

3.5.6.2 Integer data

For the case of integer data, the BADPIX value is to be a value that cannot represent valid data. The BADPIX value is not allowed to be the value 0.

3.6 Metadata

The Pilot Project IPP is to generate supplemental data known as “metadata”.

3.6.1 Metadata I/O

The generated metadata is to be written as ASCII text files on disk, associated with input image files. The disk file format shall consist of fixed-column tables and key/value pairs as appropriate. In specified instances (3.8), appropriate metadata is to be included in the output FITS file header of an image file.

3.6.2 Metadata content

For each processing module (see 3.8), generated metadata must include module version information, module exit status (pass or fail), time of execution, algorithm used by module (if any), fit statistic (if any), output data description (if any), identification of auxiliary data used (file name and last modification date, and type of input, ie flat-field image, mask data points), and, if applicable to a given module, the metadata is to include image data sky statistics.

3.7 Catalogs

3.7.1 Externally provided catalogs

Catalogs used for input to the Pilot Project IPP are to be supplied by the IfA. The catalog format is to be accepted by the IPP “as is”.

3.7.2 Object catalogs

Simple object catalogs shall be created in the form of ASCII text tables. These tables shall use the UNIX line-break convention and shall use fixed number formats for each column. Descriptive summary information shall be written in the first lines of the file. These summary lines shall begin with the hash mark (#) so that existing software will recognized these as comments.

3.8 Modules

The Pilot Project IPP design should include individual modules capable of implementing specified functionality or algorithms. Common operations/manipulations can be placed into subroutines and functions (including library functions) that can be called by a module responsible for a given algorithm, e.g., an appropriate function can be defined to compute a standard deviation without having to place the code for this computation into every routine that utilizes it. Modules shall detect error conditions and report them to the supporting engine, which shall exit with a descriptive message. Modules shall not provide defaults for input values unless specified by the algorithm documents. Modules shall return an error if the input data is not suitable.

3.8.1 Input of image data

A module shall be implemented to read image data from a disk FITS file. The public API for this module shall be of the form:

```c
readFITS (OTA *ota, char *filename);
```
where \texttt{ota} represents the data structure specifying an OTA. The function shall distinguish 2-D and 3-D image files and define the number of Cell entries in the OTA appropriately (see section 3.5.3).

### 3.8.2 Output of image data

A pair of modules shall be implemented to write image data to a disk FITS file as either a single Cell (2-D FITS file) or as a complete OTA (3-D FITS file). The public API for these modules are:

- `writeFITS_OTA (OTA *ota, char *filename, Format format);`
- `writeFITS_CELL (Cell *cell, char *filename, Format format);`

where input data \texttt{ota} and \texttt{cell} specify either the Cell or the OTA data structures and the \texttt{format} entry defines the values of BITPIX, BZERO, and BSCALE.

### 3.8.3 Subtract bias

A module shall be implemented to subtract the bias from an image already in memory. The algorithm implemented in this module is to be the algorithm specified in the ADD (PSDC-410-002). The following metadata shall be produced and written to the FITS header on image output: bias calculation method, bias parameters, module version, module date, date of execution. The public API for this module shall be:

- `debias (OTA *ota, char *method, char *stats);`

where \texttt{method} defines the bias representation method (constant, spline, polynomial) and \texttt{stats} defines the statistic applied to the overscan sample (mean, median, clipped mean). The image-dependent parameters (i.e., readnoise and overscan region) shall be determined from the image metadata. No default values for method or stats shall be assumed.

### 3.8.4 Mask bad pixels

A module shall be implemented to apply a bad pixel mask to an image. The bad pixel mask shall be specified as a mask image. The algorithm implemented in this module is to be the algorithm specified in the ADD (PSDC-410-002). The following metadata shall be produced and written to the FITS header on image output: mask image name, software version & date. The public API for this module shall be:

- `removeBadPixels (OTA *image, OTA *badpixels);`

where \texttt{image} is the image data to be masked and \texttt{badpixels} is an image containing the bad-pixel map.

### 3.8.5 Divide by flat

A module shall be implemented to divide by the flat-field image to correct for the pixel-to-pixel sensitivity variations. The flat-field image shall be supplied to the IPP in the form of an image file on disk. The algorithm implemented in this module is to be the algorithm specified in the ADD (PSDC-410-002). The following metadata shall be produced and written to the FITS header on image output: flat image name, software version & date. The public API for this module shall be:

- `divideByFlat (OTA *image, OTA *flat);`
where \textit{image} is the input image to be flattened, and \textit{flat} is the flat-field image to be applied.

### 3.8.6 Background model estimation

A module shall be implemented to determine and subtract the background model (sky emission) from an image. The background model shall include a fringe image supplied to the IPP in the form of a library of fringe templates. The algorithm implemented in this module is to be the algorithm specified in the ADD (PSDC-410-002). The following metadata shall be produced and written to the FITS header on image output: fringe image name, background model parameters, software version & date. The public API for this module shall be:

\begin{verbatim}
subtractBackground (OTA *image, OTA **fringe, int Nfringe);
\end{verbatim}

where \textit{image} is the input image to correct and \textit{fringe} represents a set of \textit{Nfringe} template fringe images.

### 3.8.7 Generation of instrument-coordinate star flux and location catalogs

A module shall be implemented to determine parameters of the objects on the image. The required object parameters are: the instrumental pixel coordinates and the total flux of the object. The measured parameters shall be written to a catalog file (3.7.3). The algorithm implemented in this module is to be the algorithm specified in the ADD (PSDC-410-002). No header metadata is specified for this module. The following catalog metadata shall be written to the catalog comment section (3.7.3): detection threshold, number of detected objects.

\begin{verbatim}
findObjects (OTA *image, char *filename, float threshold);
\end{verbatim}

where \textit{image} is the input image on which to search for the object, \textit{filename} is the output file into which the detected objects shall be written, and \textit{threshold} is the detection threshold in number of sky standard deviations for the peak above the background.

### 3.8.8 Module sequencing

The modules shall perform their actions in a sequence specified by a simple script. The script shall define the input and output data for the module and allow the input from one module to be supplied by the output of a previous module. The required sequence for the Pilot Project is: read image, bias subtract, mask, divide by flat-field, subtract sky, write image, extract objects.

#### 3.8.8.1 Simple Engine

A very basic module-sequencing engine shall be defined which uses a script to link the modules in the specified order. The engine shall manage the necessary data space for the modules.

#### 3.8.8.2 Complex Engine

For complex module sequencing, the modules shall be linked using Perl via the SWIG interfaces.
3.9 Program Data Input/Output

3.9.1 The input image shall be read from a named disk file. The image shall be read respecting the requirements of the FITS standard, including the BZERO and BSCALE keywords where appropriate.

3.9.2 Calibration files shall be read in from disk.

3.9.3 Bad pixel data shall be supplied as a FITS image on disk.

3.9.4 Metadata which is not in the FITS header will be input from a disk file.

3.9.5 The script may be modified to write out an intermediate image to disk as a FITS file for inspection.

3.9.6 The association between science image and calibration/mask data shall be provided as external metadata.

3.9.7 Test images shall be provided by IfA.

3.9.8 All point-in-time metadata entries shall be specified in Universal Time.

4.0 TEST VERIFICATION

A testing regime shall be implemented to demonstrate the working state of the provided software. Certain tests as specified shall be performed by MHPCC, with code release contingent on success. Other specified tests will be performed by IfA to verify the validity of the implemented algorithms. The tests include: software configuration tests, software integrity tests, basic unit tests, and detailed functional analysis.

4.1 Software Configuration Tests

MHPCC shall test the validity of the software configuration, specifically to check that the code can be compiled on the specified platforms and that the compilation produces no errors or warnings, except as noted and allowed.

4.2 Software Integrity Tests

MHPCC shall test the integrity of the software, specifically to check that the code does not produce memory leaks, segmentation faults.

4.3 Basic Unit Tests

MHPCC shall perform basic unit tests with sample input data and known output results, including invalid input data to test error handling. These tests should exercise the complete range of module options.

4.4 Detailed Functional Analysis

IfA shall perform detailed tests with a wide range of input data and compare the results with existing software system.
### 4.5 Verification Matrix

<table>
<thead>
<tr>
<th>Paragraph Number</th>
<th>Requirement</th>
<th>Verification Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>code via CVS</td>
<td>inspection</td>
</tr>
<tr>
<td>3.1</td>
<td>binaries via G-Forge</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.1</td>
<td>source code in C</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.2</td>
<td>scripting in Perl</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.3</td>
<td>software interface</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.4</td>
<td>ANSI Standard C99</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.5</td>
<td>coding style</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.6</td>
<td>comment conventions</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.7</td>
<td>posix compliance</td>
<td>inspection</td>
</tr>
<tr>
<td>3.2.8</td>
<td>CVS for version control</td>
<td>inspection</td>
</tr>
<tr>
<td>3.3.1</td>
<td>SDD, SRS, ADD</td>
<td>inspection</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Doxygen for commenting</td>
<td>inspection</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Man pages</td>
<td>inspection</td>
</tr>
<tr>
<td>3.4</td>
<td>architecture &amp; compilation</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.5.2</td>
<td>image size from metadata</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.5.3</td>
<td>FITS file formats</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.5.5</td>
<td>image type conversion</td>
<td>inspection</td>
</tr>
<tr>
<td>3.5.6</td>
<td>magic data values</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.6.1</td>
<td>metadata to flat-files, headers</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.6.2</td>
<td>metadata content</td>
<td>test</td>
</tr>
<tr>
<td>3.7.1</td>
<td>read in catalogs</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.7.2</td>
<td>object catalogs</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.8.1</td>
<td>input module</td>
<td>test</td>
</tr>
<tr>
<td>3.8.2</td>
<td>output modules</td>
<td>test</td>
</tr>
<tr>
<td>3.8.3</td>
<td>bias subtract module</td>
<td>test</td>
</tr>
<tr>
<td>3.8.4</td>
<td>bad pixel mask module</td>
<td>test</td>
</tr>
<tr>
<td>3.8.5</td>
<td>divide by flat module</td>
<td>test</td>
</tr>
<tr>
<td>3.8.6</td>
<td>background subtraction</td>
<td>test</td>
</tr>
<tr>
<td>3.8.7</td>
<td>object detection</td>
<td>test</td>
</tr>
<tr>
<td>3.8.8</td>
<td>module sequencing</td>
<td>demonstration</td>
</tr>
<tr>
<td>Paragraph Number</td>
<td>Requirement</td>
<td>Verification Method</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>3.9.1</td>
<td>image input [duplicate of 3.5.3]</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.9.2</td>
<td>calibration input</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.9.3</td>
<td>bad pixel input</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.9.4</td>
<td>metadata input</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.9.5</td>
<td>intermediate output images</td>
<td>demonstration</td>
</tr>
<tr>
<td>3.9.7</td>
<td>UT for time values</td>
<td>demonstration</td>
</tr>
</tbody>
</table>