

ImageJDev

Curtis Rueden, LOCI
Grant Harris, MBL at Woods Hole

Introduction

- ImageJDev: an NIH-funded project to produce the next generation of ImageJ
- Partnership between several institutions:
 - LOCI at UW-Madison
 - MBL at Woods Hole
 - Broad Institute of MIT and Harvard
 - Fiji group (MPI-CBG, Uni/ETH Zurich, etc.)

See also: imagejdev.org/collaborators

Outline

- Vision
- Aims
- Design
- Progress
- Future Directions

Outline

- **Vision**
- Aims
- Design
- Progress
- Future Directions

“Don't be pushed by your problems. Be led by your dreams.”

—Anonymous

Vision: Guiding Principles

- Preserve backwards compatibility
- Maintain good performance
- Support N-dimensional imaging
- Use common input and output for data
- Minimize complexity
 - Introduce dependencies only when benefits outweigh disadvantages
- Employ modern software development practices

Vision: The Dream

- What is ImageJ's greatest strength?

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's *extensible* by writing plugins

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's ***extensible*** by writing plugins
- How can we expand on this potential?

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's *extensible* by writing plugins
- How can we expand on this potential?
 - Plugins as *modular* “building blocks”

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's *extensible* by writing plugins
- How can we expand on this potential?
 - Plugins as *modular* “building blocks”
- What does modularity gain us?

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's ***extensible*** by writing plugins
- How can we expand on this potential?
 - Plugins as ***modular*** “building blocks”
- What does modularity gain us?
 - Modularity facilitates ***interoperability***

Vision: The Need

- **Extensibility**
- **Modularity**
- **Interoperability**

Vision: The Need

- **Extensibility**
- Modularity
- Interoperability

A system design principle where the implementation takes into consideration future growth. It is a systemic measure of the ability to extend a system and the level of effort required to implement the extension.

—“Extensibility” on Wikipedia

Vision: The Need

- Extensibility
- **Modularity**
- Interoperability

The extent to which software is composed of separate, interchangeable components, called modules, which represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components.

—“Modularity” on Wikipedia

Vision: The Need

- Extensibility
- Modularity
- **Interoperability**

The capability of different programs to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols. The lack of interoperability can be a consequence of a lack of attention to standardization during the design of a program.

—“Interoperability” on Wikipedia

Vision: The Need

- Extensibility
- Modularity
- **Interoperability**

The capability of different programs to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols. *The lack of interoperability can be a consequence of a lack of attention to standardization during the design of a program.*

—“Interoperability” on Wikipedia

Vision: The Challenge

- How do we maintain compatibility?
 - Will plugins and macros still work?
 - Do other programs work with ImageJ 2.0?

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?
 - Use standards

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?
 - Use standards
 2. Implementation: how to preserve compatibility?

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?
 - Use standards
 2. Implementation: how to preserve compatibility?
 - Small, “safe” code changes that preserve existing behavior

Vision: The Process

- Unit tests
 - A “safety net” for preserving behavior
 - The act of creating them encourages modular design
- Continuous integration
 - An “early warning system”
- Project management tools, etc....

Outline

- Vision
- **Aims**
- Design
- Progress
- Future Directions

“Goals are dreams with deadlines.”

—Diana Scharf Hunt

Aims

1. **Improve ImageJ's core architecture**
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

Aims

1. Improve ImageJ's core architecture
 - a) **Separate data model from user interface**
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) **Develop extensions framework for algorithms**
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) **Broaden the image data model**
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

Outline

- Vision
- Aims
- **Design**
- Progress
- Future Directions

“I haven't failed, I've found 10,000 ways that don't work.”

—Thomas Edison

Design

- Considered several design approaches
 - Iterative (current strategy)
 - Greenfield (new application)
 - Delegation (change IJ1's internals)
 - Adaptation (leave IJ1 alone)
- Adaptation: IJ2 includes IJ1 as a library
- IJ1 and IJ2 grow and evolve together
- More slides during roundtable if interest

Outline

- Vision
- Aims
- Design
- **Progress**
- Future Directions

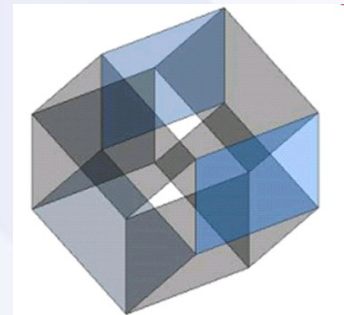


Progress

- 1) Imglib: an N-dimensional image data model
 - Bio-Formats: reading data
- 2) Automatic plugins for extensible visualization
 - Spectral lifetime image data plugin
- 3) OpenCL-based iterative deconvolution

Progress: Imglib

- Added data types backed by Imglib library
 - Currently supports nine pixel types:
 - Signed and unsigned integer, floating point
 - Bit depths: 8, 16, 32, and 64 bit
- Many possible storage strategies
- Type-independent plugins



Progress: Bio-Formats

- Adapted ImageJ to use Bio-Formats natively for reading file formats
- Files are opened as N-dimensional, imglib-backed images



Progress: Automatic Plugins

- Plugin author uses Java Annotation to label which dimensions a plugin can handle
- When image is loaded from File/Open IJ checks dimensions and finds matching plugins
- IJ automatically runs unique matching plugin, or displays dialog of choices if several match

```
@Dimensions(required="X,Y,Lifetime", optional="Channel")  
public class SLIMPlugInAuto implements IAutoDisplayPlugin {  
    ...  
}
```

Progress: Quick Demo

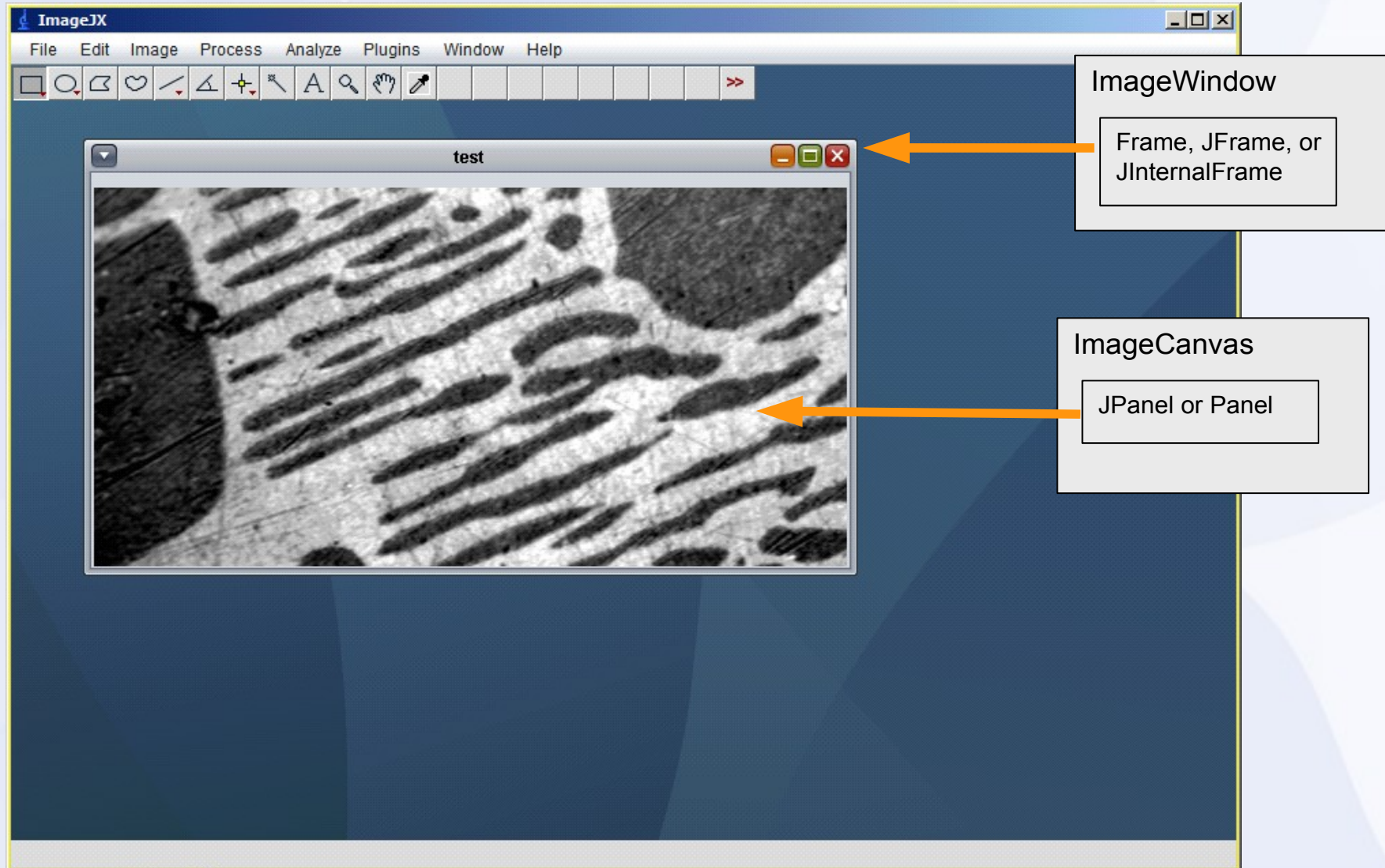
Progress: OpenCL Plugin

- OpenCL: Run software on CPU and GPU cores for fast processing-intensive analysis
- Web services: Invoke code from a remote machine—cross-language, cross-platform
- Methodology for applying iterative speed-up to existing Java code by translating to OpenCL

Progress

- 4) ImageJX: pursuing a separation of concerns
- 5) Declarative plugins for greater interoperability
 - CellProfiler connectivity with ImageJ
- 6) Requirements: community feature requests
- 7) Software development methodology and tools

Progress: ImageJX



Progress: Declarative Plugins

- Existing plugin example:

```
ImagePlus original = WindowManager.getCurrentImage();

GenericDialog gd = new GenericDialog("\"Tubeness\" Filter");
gd.addNumericField("Sigma: ",
    (calibration==null) ? 1f : minimumSeparation, 4);
gd.addMessage("(The default value for sigma " +
    "is the minimum voxel separation.)");
gd.addCheckbox("Use calibration information", calibration!=null);

gd.showDialog();
if (gd.wasCanceled()) return;

double sigma = gd.getNextNumber();
boolean useCalibration = gd.getNextBoolean();

TubenessProcessor tp = new TubenessProcessor(sigma, useCalibration);
```

Progress: Declarative Plugins

- Declarative plugin example:

```
@Parameter(label="Input image")
public ImagePlus original = null;

@Parameter(label="Sigma")
public double sigma = 1.0;

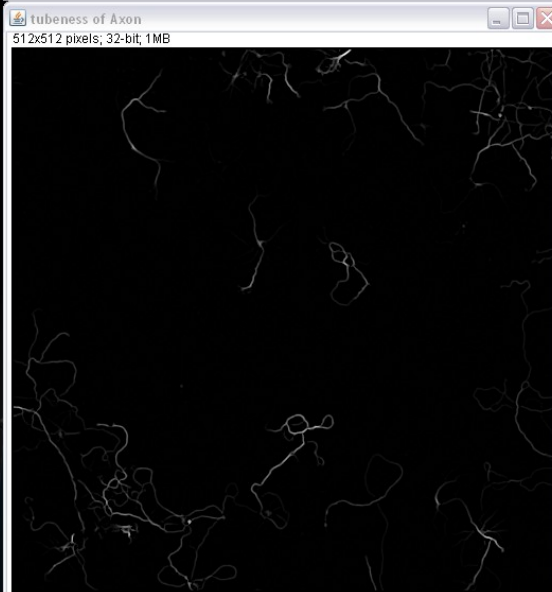
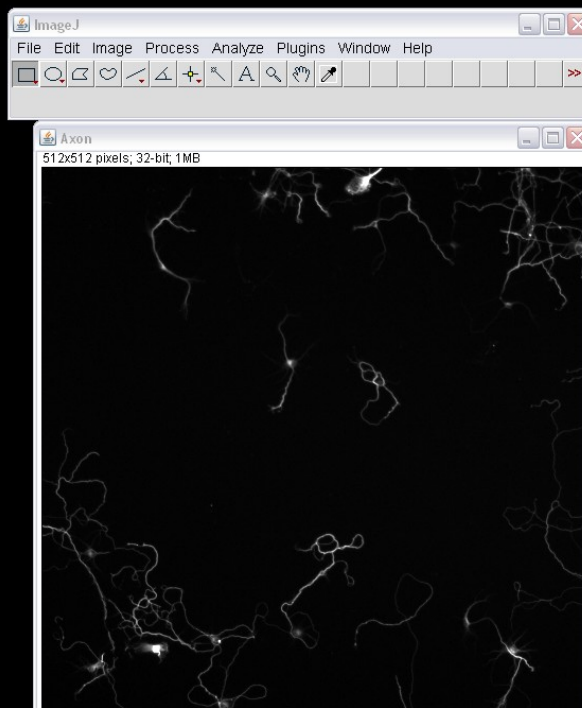
@Parameter(label="Use calibration")
public boolean useCalibration = false;

@Parameter(label="Output image", output=true)
public ImagePlus result = null;

public void run(String ignored) {
    if (original == null)
        original = WindowManager.getCurrentImage();
    TubenessProcessor tp = new TubenessProcessor(sigma, useCalibration);
    ...
}
```

Progress: CellProfiler

- CellProfiler is a tool for executing high-throughput image analysis pipelines
- Achieves better interoperability with ImageJ using the declarative plugin mechanism



CellProfiler (v.10521): Pipeline_ImageJ.cp (C:\Documents and Settings\curtis\My Documents\CellProfiler...

File Edit Test Window Data tools Help

Module notes

Command or macro? Macro

Macro: run("Tubeness ", "sigma=1.0000 use");

Set the current image? ☒

Current image: Axon

Get the current image? ☒

Final image: AxonsTubeness

Run before each group? Nothing

Run after each group? Nothing

Wait for ImageJ? ☐

Show ImageJ Show

Adjust modules: + - ^ v

PANDORA_080824200001_B01F00d0.TIF
PANDORA_080824200001_B01F00d1.TIF
PANDORA_080824200001_B01F00d2.TIF
Pipeline_ImageJ.cp
Pipeline_ImageJ2.0.cp

Default Input Folder: C:\Documents and Settings\curtis\My Docu...
Default Output Folder: C:\Documents and Settings\curtis\My Document...
Output Filename: DefaultOut_1.mat Allow overwrite? Analyze images

Welcome to CellProfiler

RunImageJ, image cycle #1

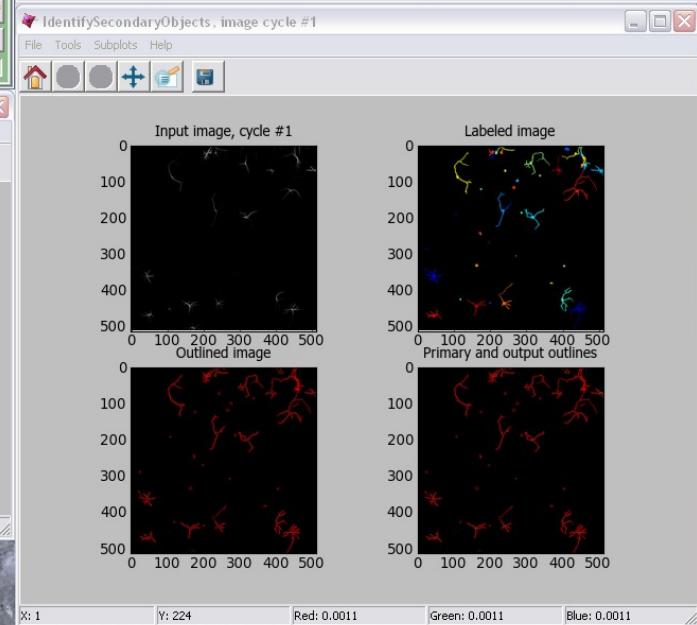
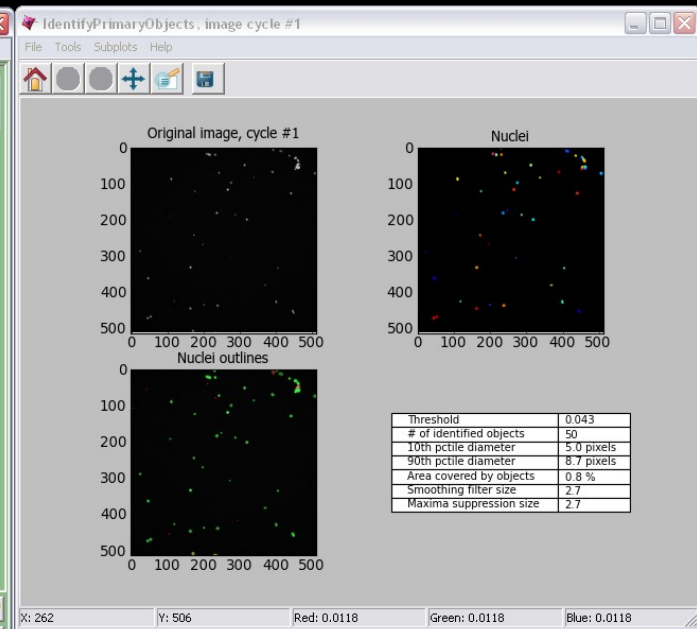
File Tools Subplots Help

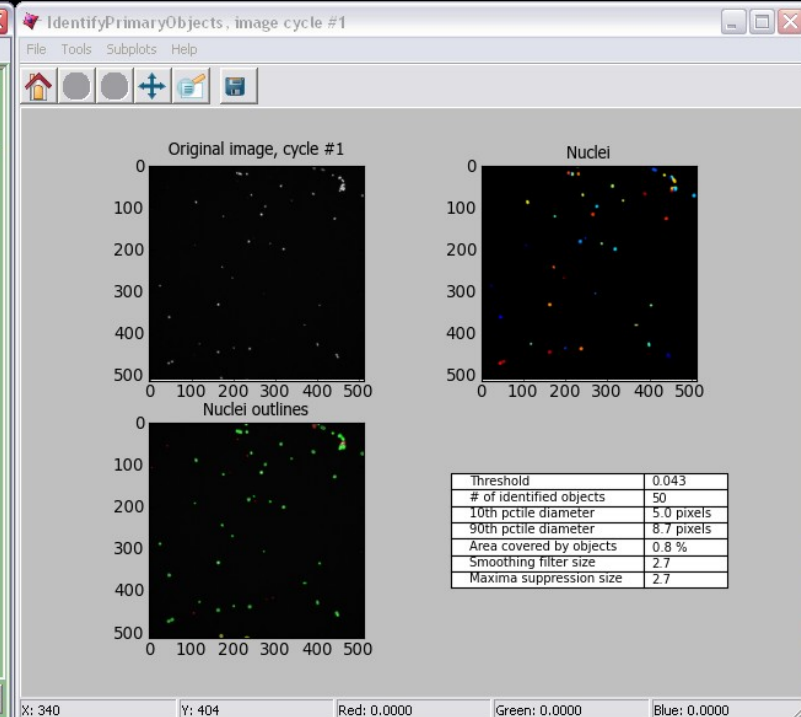
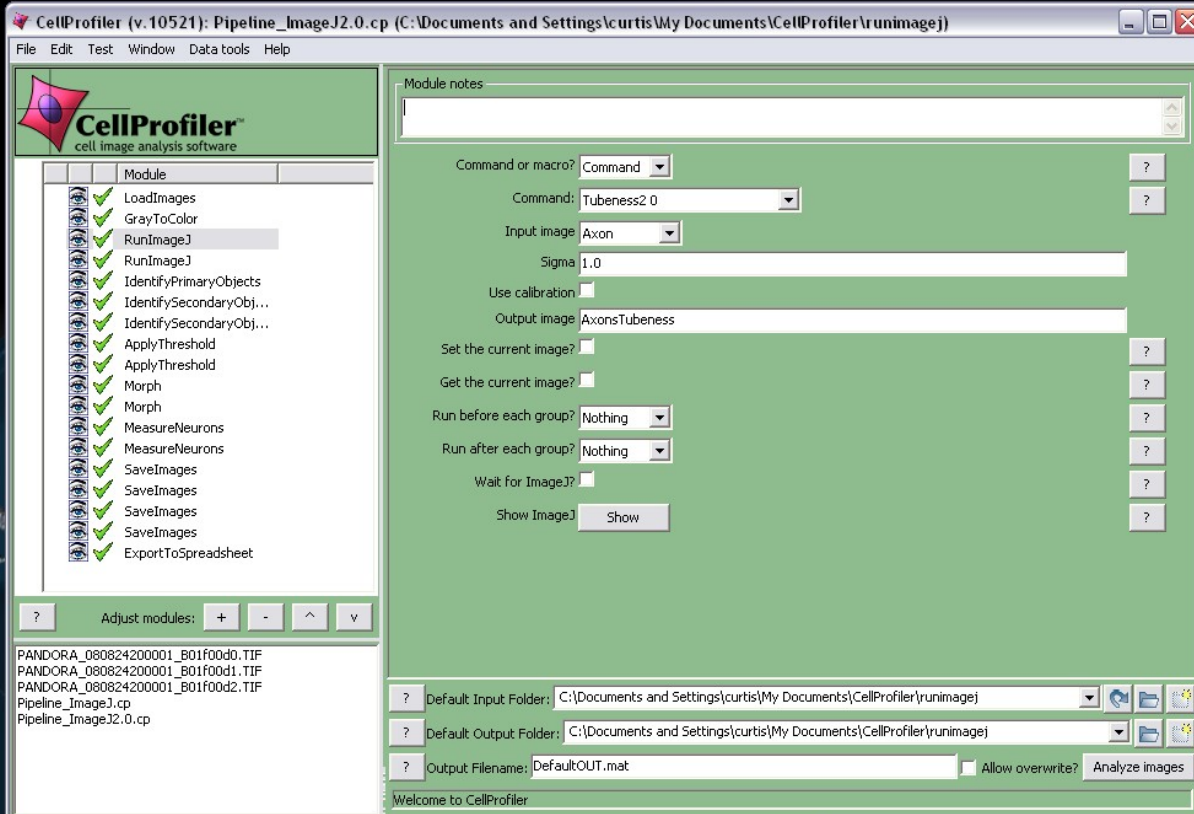
Input image: Axon

Output image: AxonsTubeness

X: 44 Y: 179 Intensity: 0.0118

Mon Oct 11 17:21:36 2010: Image # 1, module Morph # 10: 0.61 sec
Mon Oct 11 17:21:38 2010: Image # 1, module Morph # 11: 0.59 sec
Mon Oct 11 17:21:39 2010: Image # 1, module MeasureNeurons # 12: 2.88 sec (bg)
Mon Oct 11 17:21:43 2010: Image # 1, module MeasureNeurons # 13: 2.86 sec (bg)
Mon Oct 11 17:21:46 2010: Image # 1, module SaveImages # 14: 0.57 sec (bg)
Mon Oct 11 17:21:47 2010: Image # 1, module SaveImages # 15: 1.35 sec (bg)
Mon Oct 11 17:21:49 2010: Image # 1, module SaveImages # 16: 1.88 sec (bg)
Mon Oct 11 17:21:52 2010: Image # 1, module SaveImages # 17: 0.12 sec (bg)
Mon Oct 11 17:21:52 2010: Image # 1, module ExportToSpreadsheet # 18: 0.00 sec



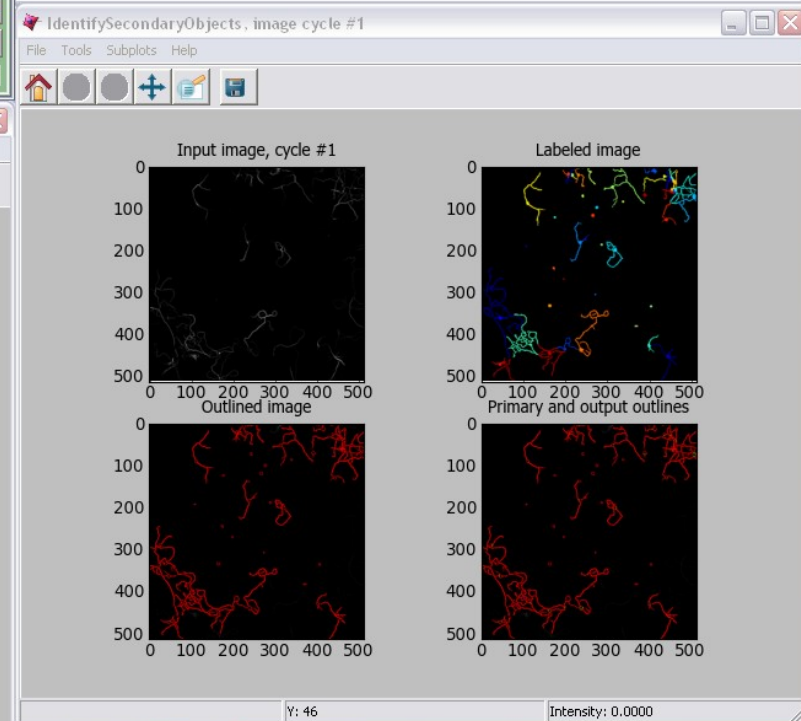
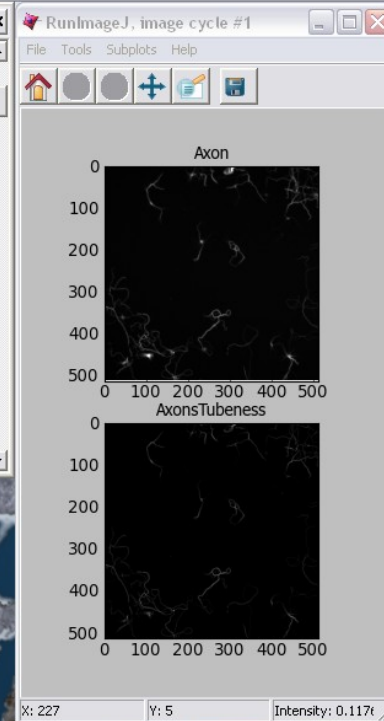


C:\Program Files\CellProfiler\CellProfiler.exe

```

Mon Oct 11 13:36:24 2010: Image # 1, module RunImageJ # 3: 2.07 sec (bg)
Mon Oct 11 13:36:27 2010: Image # 1, module RunImageJ # 4: 1.22 sec (bg)
C:\Program Files\CellProfiler\library.zip\numpy\lib\function_base.py:185: Warning
g:
The new semantics of histogram is now the default and the 'new'
keyword will be removed in NumPy 2.0.
Mon Oct 11 13:36:30 2010: Image # 1, module IdentifyPrimaryObjects # 5: 4.64 sec
(bg)
Mon Oct 11 13:36:35 2010: Image # 1, module IdentifySecondaryObjects # 6: 1.88 s
ec
Mon Oct 11 13:36:38 2010: Image # 1, module IdentifySecondaryObjects # 7: 1.89 s
ec
Mon Oct 11 13:36:41 2010: Image # 1, module ApplyThreshold # 8: 6.82 sec (bg)
Mon Oct 11 13:36:49 2010: Image # 1, module ApplyThreshold # 9: 2.79 sec (bg)
Mon Oct 11 13:36:53 2010: Image # 1, module Morph # 10: 0.59 sec
Mon Oct 11 13:36:55 2010: Image # 1, module Morph # 11: 0.61 sec
Mon Oct 11 13:36:56 2010: Image # 1, module MeasureNeurons # 12: 3.02 sec (bg)
Mon Oct 11 13:37:00 2010: Image # 1, module MeasureNeurons # 13: 2.73 sec (bg)
Mon Oct 11 13:37:03 2010: Image # 1, module SaveImages # 14: 0.56 sec (bg)
Mon Oct 11 13:37:04 2010: Image # 1, module SaveImages # 15: 0.12 sec (bg)
Mon Oct 11 13:37:05 2010: Image # 1, module SaveImages # 16: 0.10 sec (bg)
Mon Oct 11 13:37:05 2010: Image # 1, module SaveImages # 17: 0.07 sec (bg)
Mon Oct 11 13:37:06 2010: Image # 1, module ExportToSpreadsheet # 18: 0.00 sec

```



Progress: Requirements

- Gathered feedback from the community
- Major areas of ImageJ
 - Data model & image processing
 - Visualization & user interface
 - Input & output
 - Segmentation & regions of interest
 - Scripting & plugins

Progress: Development Tools

- Web site
- Unit test suite
- Continuous integration: Hudson
- Source control: Subversion & Git
- Project management: Maven & Trac

ChromeFileEditViewHistoryBookmarksWindowHelp

28°1.6KB/s0.5KB/s

07:39

ImageJAbout ImageJDev | imagejdev

imagejdev.org

BooksDocsGamesOMESciTechSocialToolsWorkMail

Other Bookmarks



imagejdev.org

Search

AboutResourcesPluginsRoadmapFor developersBlogsRecent changesLog out

About ImageJDev

ViewEditOutline



ImageJ User & Developer Conference

27-29. October 2010

ImageJDev is a **federally funded, multi-institution effort** to strengthen both the ImageJ software itself and its community by pursuing a new vision of ImageJ and associated community resources, including this website, code and plugin repositories, and user and developer documentation.

The project was launched in late September 2009, and is still in the early planning stage. This website will be updated as the project progresses.

The mission of imagejdev.org is:

- To lead ImageJ development with a clear vision.
- To continue developing one official version of ImageJ to keep the user community unified and happy.
- To collaborate with other interested parties and institutions wherever useful.
- To ensure ImageJ remains useful and relevant to the broadest possible community.
- To maintain backwards compatibility with the current ImageJ as close to 100% as possible.
- To avoid duplication of effort and instead leverage each others' work wherever practical.
- To provide a central online resource for ImageJ: program downloads, a plugin repository, developer resources and more.

See the menu on the right for more information.

Attachment	Size
imagej-conference-2010.jpg	34.1 KB

Primary links

About

- Aims
- Collaborators
- Funding
- History
- Proposal
- Rationale

Resources

Plugins

Roadmap

For developers

Blogs

Recent changes

Log out

curtis

My account

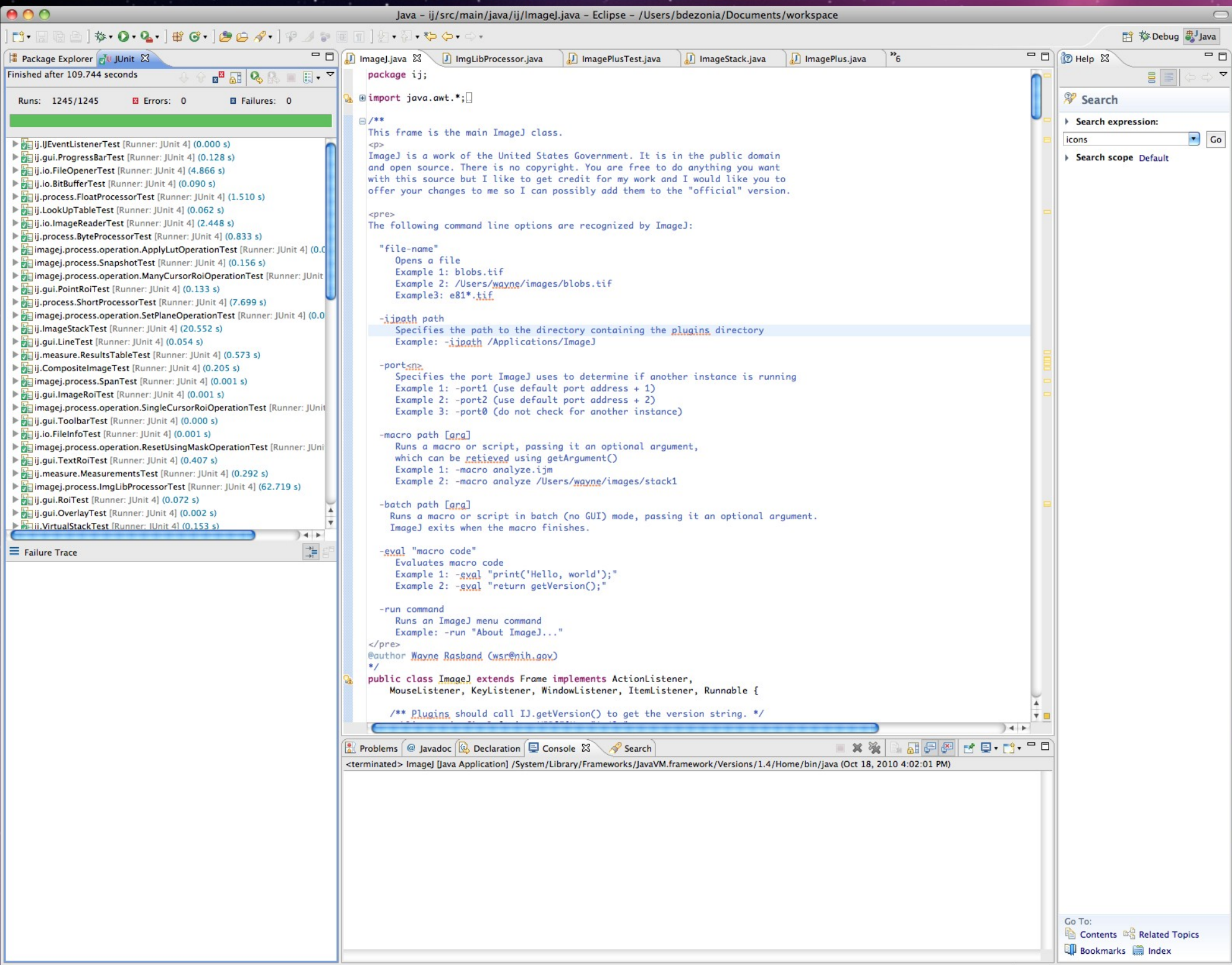
Recent changes

Web Links

Create content

Feed aggregator

Log out



Hudson

search ? log in

Hudson

DISABLE AUTO REFRESH

People

Build History

Build Queue

No builds in the queue.

Build Executor Status

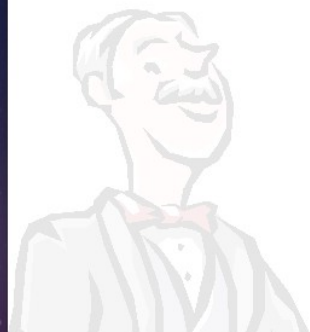
#	Status	
1	Idle	
2	Idle	

All

S	W	Job ↓	Last Success	Last Failure	Last Duration
		ImageJ	9 hr 11 min (#353)	21 hr (#350)	4 min 23 sec
		LOCI	9 hr 11 min (#299)	4 days 14 hr (#289)	2 min 31 sec

Icon: [S](#) [M](#) [L](#)

[Legend](#)  [for all](#)  [for failures](#)  [for just latest builds](#)




Timeline - ImageJ

About ImageJDev | imagejdev

dev.imagejdev.org/trac/imagej/timeline

Books Docs Games OME SciTech Social Tools Work Mail

Other Bookmarks



[Login](#)
[Preferences](#)
[Help/Guide](#)
[About Trac](#)
[Forgot your password?](#)

[Wiki](#)
[Timeline](#)
[Roadmap](#)
[Browse Source](#)
[View Tickets](#)
[Search](#)

[Previous Period](#)
[Next Period](#)

Timeline

10/18/10: Yesterday

15:33

Changeset [\[1582\]](#) by aivar

Save, set and restore WindowManager's tempCurrentImage.

15:32

Changeset [\[1581\]](#) by aivar

Don't call AutoPluginInvoker when running a macro (it may popup a dialog).

13:31

Changeset [\[1580\]](#) by curtis

Fix Maven configuration issue with sezpoz library.

13:31

Changeset [\[1579\]](#) by curtis

Fix Maven configuration issue with sezpoz library.

11:46

Changeset [\[1578\]](#) by aivar

Sets temporary image in WindowManager before loading the plugin.

11:45

Changeset [\[1577\]](#) by aivar

Added sezpos dependency.

11:19

Ticket [#147](#) (Bioformats can't open Metamorph z-stack) closed by leek

duplicate: This is being addressed by other work on the 5-channel representation.

10:42

Ticket [#260](#) (Add way to plugins automatically) created by aivar

Add annotations for required and optional dimensions, provide interface ...

10:27

Ticket [#250](#) (Try to merge imagej ijk branch back to trunk) closed by curtis

Invalid: The two branches are not yet ready to be merged. Regardless, with our ...

10/15/10:

23:00

Milestone [biweekly-2010: Oct-04 to Oct-15](#) completed

Tasks from 2010-Oct-04 through 2010-Oct-15.

View changes from

10/19/10

and

30

days back

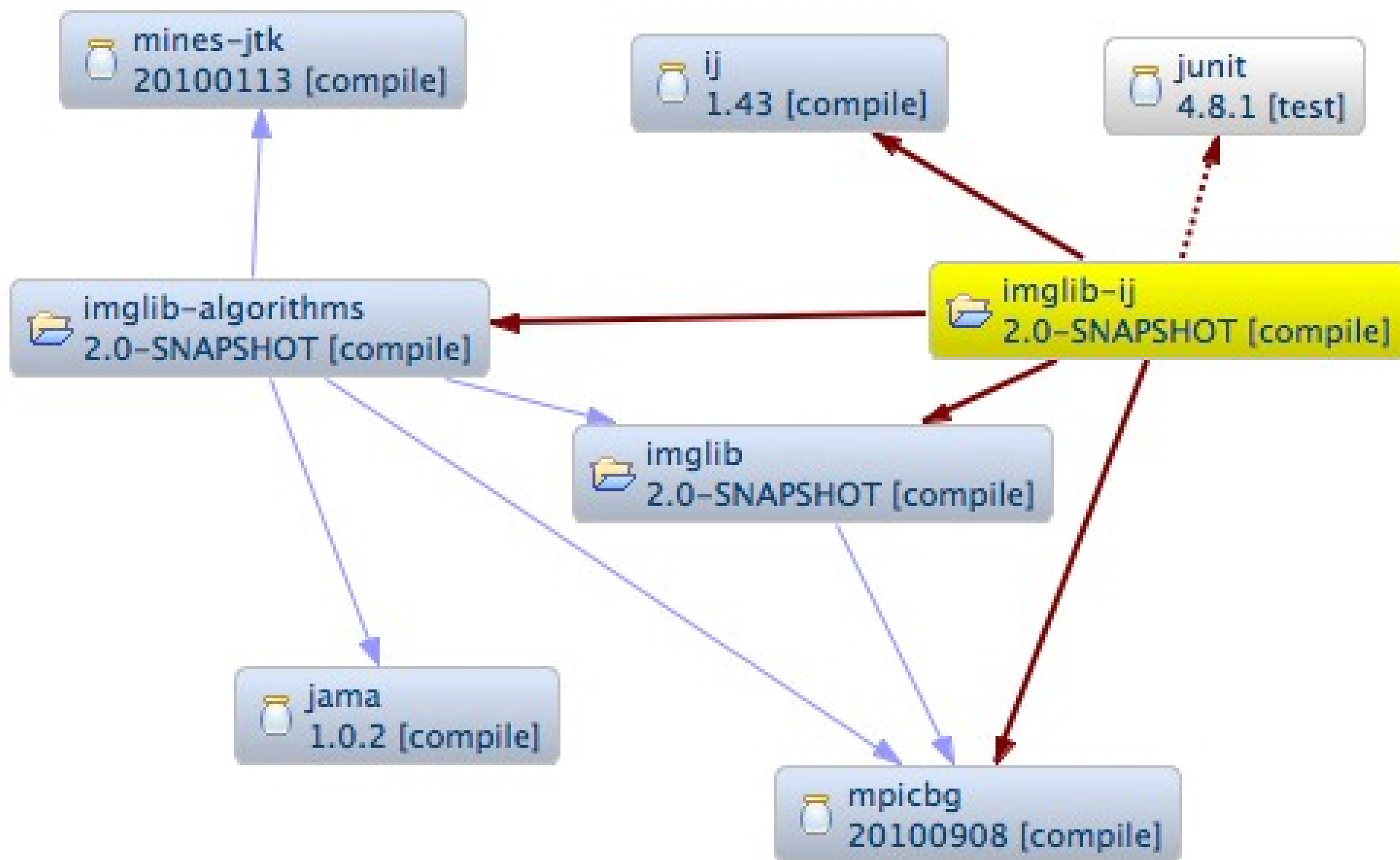
done by

☒ Opened and closed tickets
 ☐ Ticket updates
 ☒ Repository checkins
 ☒ Milestones
 ☒ Wiki changes

Update

Dependency Graph [test]

Search:



ImageJ

Last Published: 2010-10-19

ImageJ

Parent Project

ImageJ Base POM

Project Documentation

Project Information

About

Continuous

Integration

Dependencies

Dependency

Convergence

Issue Tracking

Mailing Lists

Plugin Management

Project License

Project Plugins

Project Summary

Project Team

Source Repository

Project Reports



Reactor Dependency Convergence

Legend:

- ▲ All projects share one version of the dependency.
- ✖ At least one project has a differing version of the dependency.

Statistics:

Number of sub-projects:	4
Number of dependencies (NOD):	8
Number of unique artifacts (NOA):	8
Number of SNAPSHOT artifacts (NOS):	3
Convergence (NOD/NOA):	▲ 100%
Ready for Release (100% Convergence and no SNASHOTS):	✖ Error You have SNAPSHOT dependencies.

Dependencies used in sub-projects

com.sun:tools

▲	1.4.2	a. imagej:ij
---	-------	------------------------------

imagej:bf-imglib

Outline

- Vision
- Aims
- Design
- Progress
- **Future Directions**

“Your task is not to foresee
the future, but to enable it.”
—Antoine de Saint Exupéry

Future Directions

- Pursue Adaptation design for IJ 2.0
- N-dimensional image data model
- Investigate standards useful to ImageJ
 - Rich client platform for user interface
 - Modularity and interoperability: e.g., OSGi
 - ROIs: e.g., JHotDraw
- Improve headless behavior
- Implement community requirements

Summary

- What Will ImageJ 2.0 Do for Me?
 - Work with existing plugins and macros
 - Work with new, exciting plugins and scripts
 - Handle larger, more complex datasets
 - Multidimensional visualization tools
 - Easier to link with other software
 - Easier plugin management

Acknowledgements

- **Principal Investigators**

- Kevin Eliceiri (LOCI), Rudolf Oldenbourg (MBL), Anne Carpenter (Broad)

- **Developers**

- Grant Harris, Barry DeZonia, Aivar Grislis, Rick Lentz (ImageJDev)
- Lee Kametsky, Adam Fraser (CellProfiler)

- **Collaborators**

- Wayne Rasband (ImageJ)
- Pavel Tomancak, Johannes Schindelin, Albert Cardona (Fiji)
- Stephan Preibisch, Stephan Saalfeld (Imglib, Fiji)
- Mark Longair, Jean-Yves Tinevez (Fiji)
- Jason Swedlow, OMERO development team (OME)

Discussion

- Comments? Questions?
- Thoughts on what ImageJ 2.0 should be?
- Ideas from the community

Design Approaches

Design Approaches

1. Iterative

- Pro: No project forks
- Pro: Maintains compatibility whenever possible
- Pro: Brings code “under test”
- Con: Heavily constrained by the existing design
- Con: Development is slow

2. Greenfield

- Pro: Great flexibility
- Pro: Rapid development
- Pro: New code is “under test”
- Con: No compatibility
- Con: Forks the project
- Con: Loses legacy codebase's “embedded knowledge”

Design Approaches

1. Iterative

- Pro: No project forks
- Pro: Maintains compatibility whenever possible
- Pro: Brings code “under test”
- Con: Heavily constrained by the existing design
- Con: Development is slow

2. Greenfield

- Pro: Great flexibility
- Pro: Rapid development
- Pro: New code is “under test”
- Con: No compatibility
- Con: Forks the project
- Con: Loses legacy codebase's “embedded knowledge”

Design Approaches

1. Iterative + 2. Greenfield

?

Design Approaches

Approach #3: Delegation

- Good compatibility
- Good design flexibility
- But very disruptive of legacy work

Design Approaches

Approach #4: Adaptation

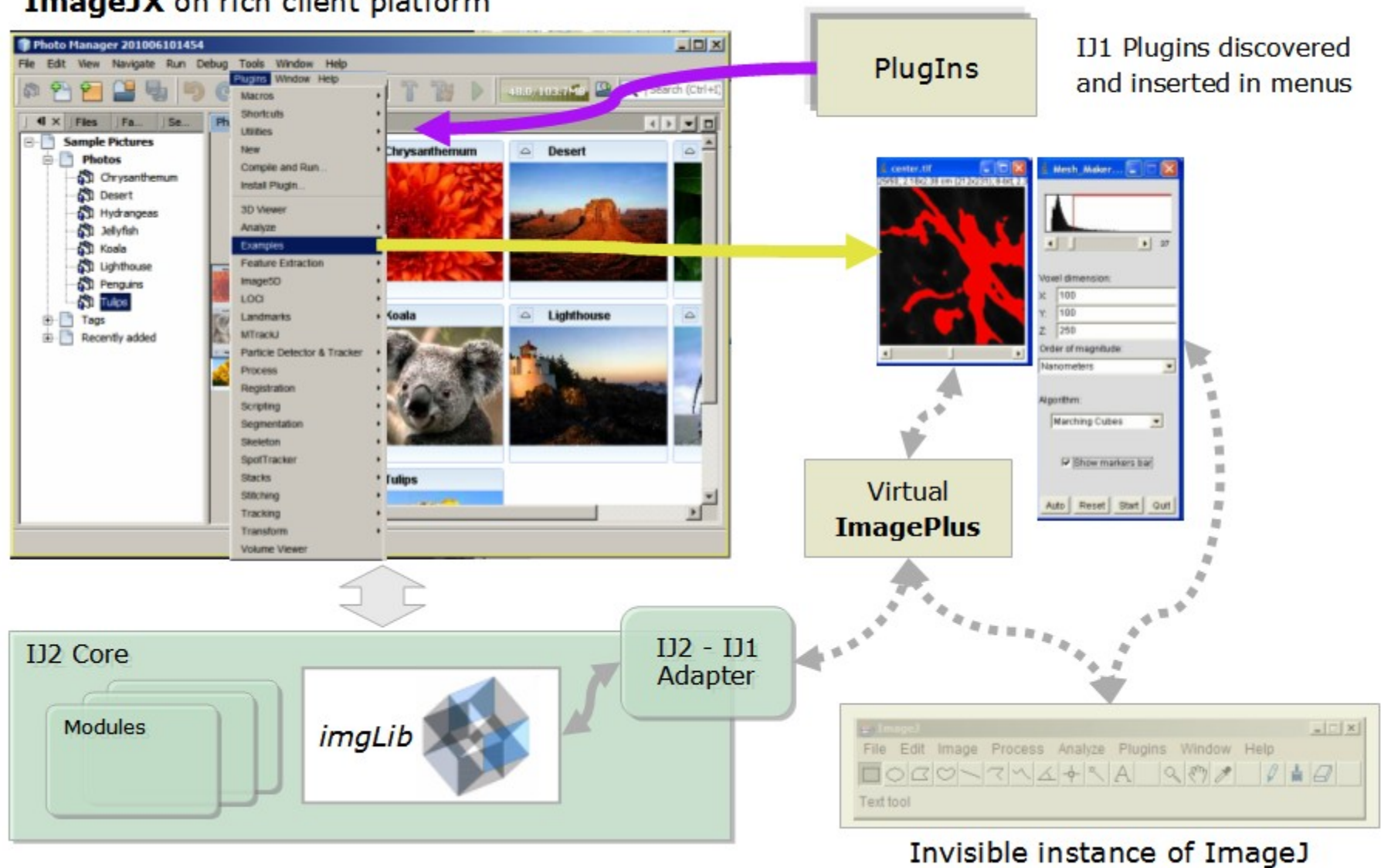
- Nearly perfect compatibility
- Smooth transition from legacy code
 - Legacy work continues as long as needed

Design Approaches

Approach #4: Adaptation

- Some limits to interoperability
- Harnesses “embedded knowledge” of legacy work without being constrained by it

ImageJX on rich client platform



Community Use Cases

Use Cases: VisBio

- Limited support for large datasets
 - Image planes larger than 2GB
 - Datasets larger than available RAM
 - VirtualStacks cache only one plane at a time
- No support for 3D visualization
 - Volume rendering
 - Arbitrary slicing
 - Realtime animation
- Also needs better support for ROIs

Use Cases: Slim Plotter

- No support for new dimensions
 - Emission spectra
 - Lifetime
 - Polarization
- No support for processing inherent to viz
 - Exponential curve fitting
 - Spectral unmixing

Use Cases: Fiji

- Distributing plugins is external to ImageJ
- Keeping everything up to date is complex
- No standard for documenting plugins
- Not easy enough to prototype algorithms
 - Plugins require too much boilerplate code
 - No modular command framework for using Macro Recorder with scripts
 - Case logic for multiple pixel types is messy
- AWT dependencies preclude headless use

Use Cases: TrakEM2

- No support for displaying registered images
 - No display mechanism for multiple image tiles
 - No mechanism for transformation from data to display (e.g., affine)
- Regions of interest are limited
 - No vector-based ROIs (i.e., ROIs are bitmasks)
 - Multiple ROIs are tacked on (ROI Manager)
 - Confusing interplay between ROIs, masks & thresholds with measurement tools

Use Cases: ROIs (Michael Doube)

- Recently I've been frustrated by ROI's being limited to 2D. With the emerging utility of the 3D viewer and the proposal that ImageJ 2.0 handles N-dimensional data, it makes sense that ROIs should keep up with this development.
- In other words, in an N-dimensional image, one should be able to specify and visualise an N-dimensional ROI. So you can have a 3D VOI, and a 4D VOI with time limits (or even changing shape over time), or limit the ROI to a channel (5D).

Use Cases: ROIs (J-Y Tinevez)

- I recently tried to code weird shapes as ROIs in ImageJ. They were the results of a segmentation with constrained shapes. Because I wanted to have something nice for the user, The ROIs had to be mouse-interactive (resizable, moveable etc..). I had a difficult time.
- Johannes proposed on the Fiji-devel list an abstract class whose goal was to facilitate this interaction.
- But we still gave to comply to ImageJ `ij.gui.Roi` master class, which is a concrete class in charge of drawing rectangle ROIs. Inside this class, there is everything: the logic to draw it, to interact with the user, with the image container, and the image data. Any homemade ROI must inherit from this class, there is no interface to implement.

Use Cases: ROIs (J-Y Tinevez)

- What I would like to propose here is to go for an interface hierarchy for ROIs, that is well decoupled, and that would allow the flexible design of new ROIs.
- We use ROIs for many purposes, for instance:
 - user interaction
 - draw a rectangle to crop an image
 - measure intensity with a complex area
 - add non-destructive annotations
 - as input/output for plugins, for instance a result of segmentation
- From this you can see that they need to:
 - know how to draw themselves as an overlay
 - comply to some interface to be an input of some plugins
 - know how to interact with mouse clicks and drag

Use Cases: μ Manager (N. Stuurman)

- 1. The Brightness/Contrast tool. Display of the histogram cannot be reliably set to the dynamic range of the camera (i.e., it always automatically goes back to the range of the minimum and maximum pixel value in the image, which can be extremely deceptive). No gamma correction. No method to update histogram when the image changes. No log display of the histogram. We ended up writing our own, but things are still clunky because acquired images (shown in a modified Image5D viewer) can only be controlled by the ImageJ B&C tool.

Use Cases: μ Manager (N. Stuurman)

- 2. Lack of plugin API. We have been bitten a number of times by internal changes in ImageJ breaking our code. Wayne is very responsive, but this still causes confusion.
- 3. Lack of standard for Multi-Dimensional viewer. We ended up using Image5D viewer, Hyperstacks came later. My impression is that the UI of Image5D is easier for users than the UI of Hyperstacks. In any case, we will be helped by a standard viewer for multi-dimensional images that integrates nicely with other ImageJ tools (like 3D viewers), and that is extensible (we do need to add a number of buttons that interface with image acquisition).

Use Cases: μ Manager (N. Stuurman)

- 4. MDI versus SDI. Not sure if this was on your list already (all of you have certainly debated this in the past!), but it seems that many people prefer the MDI model. On the Mac, it is pretty weird that a single application has different menus depending on which window you select (in our case, ImageJ windows versus Micro-manager window).

Use Cases: Miscellaneous

- G. Landini: no color space support (e.g., HSB)
- F. Hessman: domain coordinate systems
 - S&S are planning support within imglib
 - ImageJX consensus is to punt on this for now
 - Need to find a group with this use case first
- Legacy AWT interface limits use of Swing
 - ImageJ cannot use different L&Fs
 - AWT is missing features (JSpinner, JInternalPane)
 - Swing development is active, unlike legacy AWT

Use Cases: Compatibility

- Advantage of ImageJ: wealth of existing code
- Problem: ImageJ2 will break that code
- Examples:
 - `ImageProcessor.getPixels()`
 - All non-private, non-final fields
 - Subclasses created to sidestep API issues
 - Even private fields—`setAccessible(true)`

Use Cases: Interoperability

- FARSIGHT: ITK-driven segmentation routines are difficult to use from Java
- CellProfiler: How can scientists combine workflows between CellProfiler and ImageJ?
- OMERO: Database-backed images are kludgy
- Others: KNIME, Endrov, BioImageXD, PSLID...

Use Cases: Performance

- Traditional tradeoff between space & time
- Tradeoff between generality & performance
 - Moving toward generality requires that we remain aware of performance issues
 - But flexibility and usability remain paramount
- OpenCL is promising but negates many of imglib's gains in generality

Components of ImageJ2

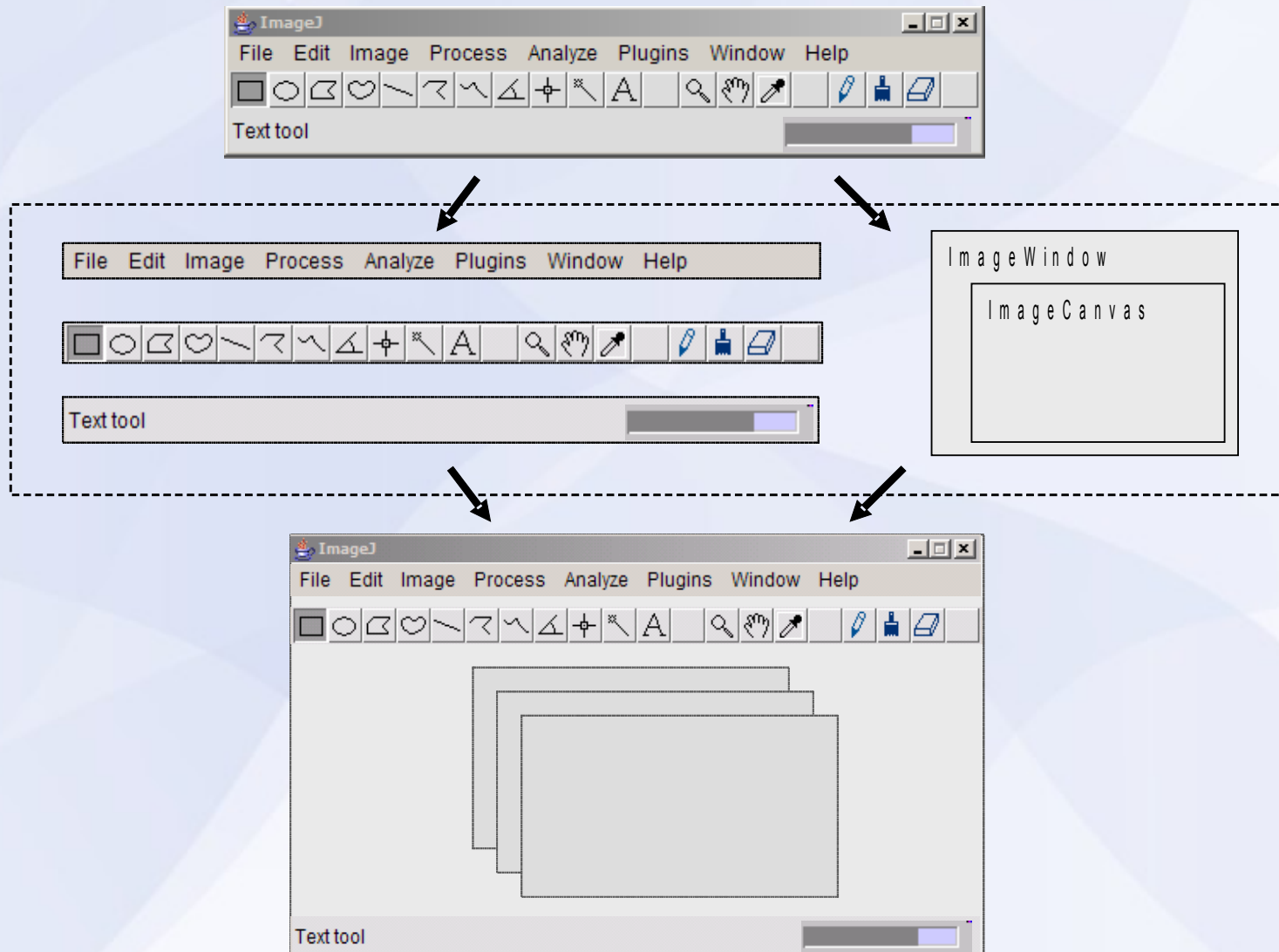
- Relevant technologies
 - 1) **Data model** – imglib library
 - 2) **Display** – Java AWT, JAI, Swing, RCP
 - 3) **Input/output** – Bio-Formats architecture
 - 4) **Regions of interest** – Java AWT, JHotDraw, OME-XML
 - 5) **Scripting & plugins** – Java 6 Scripting Framework
- More exploration of some technologies needed

ImageJX: Separation of Concerns

Decouple GUI dependencies

- Alternative GUI configurations (e.g., Swing SDI/MDI)
- Headless operation
- Incorporation into application framework
- Easing use as a library

GUI Decoupling



Dynamic Plugin Discovery

- Declarative Registration using Annotations
 - Menus, etc., are built dynamically from plugin declarations
- Classes do not need to be loading
 - Uses ‘compile-time caching’ (SezPoz)
- ‘Automatic Plugins’
 - I/O (Bio-Formats reader)
 - Display—invoke a plugin in response to a particular kind of data being opened

Dynamic Plugin Discovery

```
package demo.plugin1;

import demo.api.MenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

@MenuItem(
    label = "Exit",
    menu = "File",
    icon = "demo/plugin1/movieNew24.gif",
    bundle = "demo.plugin1.properties")

public class ExitAction implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

Toward Modularity & Extensibility

- Use interfaces, abstract classes, factories
 - Replaceable implementations
 - Enables dynamic assembly
- @ServiceProvider (e.g. SavePrefs)
- CentralLookup
- 'Injectable Singletons'
- EventBus
- Context / Selection management

ImageJDev

Curtis Rueden, LOCI
Grant Harris, MBL at Woods Hole

1

Thanks for the opportunity to speak to you all, and to Andreas for inviting me. My name is Curtis Rueden of the Laboratory for Optical and Computational Instrumentation. Grant Harris of the Marine Biological Laboratory at Woods Hole unfortunately could not be here in person due to a personal injury, but is listening in via Skype, and will be available during the round table discussion afterwards.

Introduction

- ImageJDev: an NIH-funded project to produce the next generation of ImageJ
- Partnership between several institutions:
 - LOCI at UW-Madison
 - MBL at Woods Hole
 - Broad Institute of MIT and Harvard
 - Fiji group (MPI-CBG, Uni/ETH Zurich, etc.)

See also: imagejdev.org/collaborators

2

The ImageJDev project seeks to create the next generation version of ImageJ. We'll describe what we mean by that shortly, but first some background on who we are.

ImageJDev is a collaboration between several institutions: 1) LOCI, which is a biophotonics lab in Madison, Wisconsin, USA; 2) MBL at Woods Hole in Massachusetts, an international center for research, education, and training in biology, biomedicine and ecology; 3) the Broad Institute in Boston, a cross-disciplinary group researching systematic approaches to biological sciences; 4) the Fiji group, consisting of several different institutions; and 5) Wayne Rasband, the author of ImageJ.

There are four full-time developers at LOCI including myself, Grant Harris at Woods Hole, two developers at the Broad Institute focusing on CellProfiler integration, and several other contributors and advisors including Wayne Rasband, the Fiji developers and members of the ImageJX mailing list.

See the web site for a complete list of collaborators.

Outline

- Vision
- Aims
- Design
- Progress
- Future Directions

3

This talk will describe the ImageJDev effort, including our vision and goals for ImageJ 2.0; proposed design of the software; progress so far; and what's coming, both over the next year and longer term.

Please feel free to interrupt with simple questions during the presentation. For extended discussion of more complex issues, please make a note and bring it up during the round table discussion.

Outline

- **Vision**
- Aims
- Design
- Progress
- Future Directions

“Don't be pushed by your problems. Be led by your dreams.”

—Anonymous

It is important to spend some time discussing the vision of the project, and the rationale behind it.

Vision: Guiding Principles

- Preserve backwards compatibility
- Maintain good performance
- Support N-dimensional imaging
- Use common input and output for data
- Minimize complexity
 - Introduce dependencies only when benefits outweigh disadvantages
- Employ modern software development practices

5

First, let's define some guiding principles, tenets we will follow as we

As development continues, our specific approach may change, but there are rules we won't break

Vision: The Dream

- What is ImageJ's greatest strength?

6

Now that we have some principles to ground us, let's take a moment to ponder: what is ImageJ's greatest strength?

Of course, there are many possible answers—its simplicity, fast performance, large community of users—but we would say its greatest strength is its extensibility.

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's **extensible** by writing plugins

7

You can extend ImageJ's ability to perform image processing by creating plugins, macros and scripts. It's a powerful technique, but also easy to do—and surely one of the primary reasons for ImageJ's success.

Combined with the fact that the software is open source, this extensibility has enabled ImageJ to become a community-driven phenomenon.

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's **extensible** by writing plugins
- How can we expand on this potential?

With that in mind, is there a way we can take it one step further? Can we take this potential for extensibility and make it even better, without compromising ImageJ's many other strengths?

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's **extensible** by writing plugins
- How can we expand on this potential?
 - Plugins as **modular** “building blocks”

9

Well, if it were not only easy to *write* a plugin, but also easy for others to *reuse* your plugin... we would have an ever-increasing collection of “building blocks” to choose from—a collaborative, modular design.

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's **extensible** by writing plugins
- How can we expand on this potential?
 - Plugins as **modular** “building blocks”
- What does modularity gain us?

10

Such a modular system provides building blocks for use not only within ImageJ itself, but also from other software systems.

Vision: The Dream

- What is ImageJ's greatest strength?
 - It's ***extensible*** by writing plugins
- How can we expand on this potential?
 - Plugins as ***modular*** “building blocks”
- What does modularity gain us?
 - Modularity facilitates ***interoperability***

11

As such, modular components provide the means for other software to *interoperate* with ImageJ, since each module can be used individually, overridden or swapped out, like parts under the hood of a car.

Vision: The Need

- Extensibility
- Modularity
- Interoperability

12

To summarize, we can pursue our dream of strengthening ImageJ by improving ImageJ's *extensibility*, its *modularity* and its *interoperability*. Let's briefly examine what each of these terms means.

Vision: The Need

- **Extensibility**
- Modularity
- Interoperability

A system design principle where the implementation takes into consideration future growth. It is a systemic measure of the ability to extend a system and the level of effort required to implement the extension.

—“Extensibility” on Wikipedia

13

First of all, better extensibility will make it easier than ever to write plugins and scripts, build on each others' work, and expand ImageJ's capabilities in all sorts of ways.

As the old programming proverb goes, the system should “make simple things easy, and difficult things possible.”

Vision: The Need

- Extensibility
- **Modularity**
- Interoperability

The extent to which software is composed of separate, interchangeable components, called modules, which represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components.

—“Modularity” on Wikipedia

14

Secondly, a modular design makes ImageJ easier to understand by dividing what the program can do into clear component parts. And it will make both extensibility and interoperability much more achievable.

First and foremost, ImageJ must provide the tools for building these modules—a “system for extending the system,” if you will.

Beyond that, it should provide the core modules for scientific image processing. As software developers create additional modules of common utility, they should become part of the standard ImageJ distribution.

Vision: The Need

- Extensibility
- Modularity
- **Interoperability**

The capability of different programs to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols. The lack of interoperability can be a consequence of a lack of attention to standardization during the design of a program.

—"Interoperability" on Wikipedia

15

Lastly and perhaps most importantly, ImageJ must interoperate with other software in order to be useful.

At LOCI, interoperability is our mantra. It is a central goal of everything we do, and in a broader sense, a goal of science as a whole. For those of you familiar with the Bio-Formats library and the Open Microscopy Environment consortium, these tools were designed at every level for use with other software.

We want ImageJ to be similarly flexible.

Vision: The Need

- Extensibility
- Modularity
- **Interoperability**

The capability of different programs to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols. *The lack of interoperability can be a consequence of a lack of attention to standardization during the design of a program.*

—“Interoperability” on Wikipedia

16

Note the second half of this definition, taken from Wikipedia: “The lack of interoperability can be a consequence of a lack of attention to *standardization*.”

The essence of interoperability is the use of standards: data structures or communication protocols common to multiple programs. Hence, to achieve true interoperability, we must leverage existing approaches whenever possible—and define our own when nothing suitable already exists.

Of course, such tools and standards must be chosen selectively and cautiously. But there is great benefit to doing so.

Vision: The Challenge

- How do we maintain compatibility?
 - Will plugins and macros still work?
 - Do other programs work with ImageJ 2.0?

17

That said, there are still challenges.

Of particular note is this: to improve a program, we must change it. Unfortunately, changing code is inherently dangerous, because it is extremely fragile; changing a single character can transform a working program into non-functional junk—or worse, have subtle, far-reaching consequences on ostensibly unrelated parts of the program.

Hence, one primary challenge is maintaining compatibility with the wealth of existing plugins, macros, scripts and other software.

Fortunately, there are solutions.

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?

18

So, we have identified a need for interoperability and extensibility.

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?
 - Use standards

19

And we've noted that using existing standards provides a path toward achieving those goals.

However, while the use of standards contributes much toward good software *design*, it often ignores the issue of good software *implementation practices*...

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?
 - Use standards
 2. Implementation: how to preserve compatibility?

20

...and doesn't help answer our second question: how do we maintain compatibility with existing software?

Hence, standards may seem somewhat analogous to a building's blueprint: they describe the final product, but not the physical process of construction.

However, it turns out there *are* standards in the community for software development processes as well. And just as we benefit from utilizing standard software libraries and formats, we can also take advantage of these standard processes.

Vision: The Solution

- Two primary questions:
 1. Planning: how to achieve interoperability, modularity and extensibility?
 - Use standards
 2. Implementation: how to preserve compatibility?
 - Small, “safe” code changes that preserve existing behavior

21

Specifically, good software development consists of a methodology that stresses small code changes, with verification at every step that the program's behavior is maintained.

Vision: The Process

- Unit tests
 - A “safety net” for preserving behavior
 - The act of creating them encourages modular design
- Continuous integration
 - An “early warning system”
- Project management tools, etc....

22

Further, by creating a collection of automated routines called *unit tests* that verify each existing individual program function remains unchanged, we can have a safety net for determining whether a given change has harmed compatibility.

Going a step further, we can employ a continuous integration system to automatically run these tests every time someone makes a change—and if any tests fail, email the offender about it. This catches any problems introduced as early as possible.

Such details on software development processes have filled many books, so I'll stop there, but hopefully you get the idea that standards help here as well.

Outline

- Vision
- **Aims**
- Design
- Progress
- Future Directions

“Goals are dreams with deadlines.”
—Diana Scharf Hunt

23

So, now that you know **why** we are doing this, we'll briefly describe **what** we are funded to do. We have defined three major project aims, related to our vision for ImageJ2.

Aims

1. **Improve ImageJ's core architecture**
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

24

Aim 1 is focused on reengineering ImageJ to target the goals we just described. In essence, Aim 1A is about modularity, Aim 1B targets extensibility, and Aim 1C improves interoperability.

Aims

1. Improve ImageJ's core architecture
 - a) **Separate data model from user interface**
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

25

Aim 1A focuses on separating ImageJ's central processing logic from its user interface. We want it to be possible to execute plugins, macros and other processing tasks without requiring any user interaction or displaying any windows. This idea is known as “separation of concerns” and is very related to the concept of modularity we just described.

By respecting this separation of concerns, many new things become possible. It becomes easier to run ImageJ on a cluster, or as a client-server application. It eliminates the dependency on any particular user interface, so for example the ImageJ2 interface could use Swing or SWT instead of AWT, enabling many more standard interface features such as additional widgets, multiple document interface layouts, and window docking. And it becomes much easier to develop a version of ImageJ for mobile devices or the web.

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) **Develop extensions framework for algorithms**
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

26

Aim 1B is focused on improving ImageJ's plugin mechanism. As we discussed earlier, ImageJ's extensibility is one of its key strengths, and by improving how plugins work, we make ImageJ more powerful and easier to use for science.

We'll show some examples of these improvements later, including declarative plugins, display plugins, and a metadata-rich plugin discovery mechanism.

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) **Broaden the image data model**
2. Expand interoperability with other tools
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

27

Aim 1C refers to the addition of several key features to ImageJ's processing capabilities: datasets beyond five dimensions, very high resolution image planes, data stored remotely, and a richer set of supported metadata.

Our main approach for accomplishing this sub-aim is to use an imaging library called imglib, developed at MPI-CBG in Dresden. Imglib is currently part of Fiji, but is only partially compatible with ImageJ.

Later, we'll show our progress integrating imglib with ImageJ to enable true N-dimensionality, more types of images, and flexible sources of data. We'll also show an example of higher-dimensional data: a plugin for working with combined spectral lifetime images.

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. **Expand interoperability with other tools**
3. Grow ImageJ community resources

See also: imagejdev.org/proposal

28

Aim 2 seeks to connect ImageJ with other existing imaging tools. This work will help development proceed in a practical direction with an emphasis on interoperability. By doing this, we will ensure that the improvements from Aim 1 are not being done in a vacuum, but rather with specific use cases in mind.

While Aim 2 focuses on two specific tools, CellProfiler and VisBio, our goal is really to integrate the various use cases suggested by the community. CellProfiler and VisBio are highlighted in the proposal because they represent two opposite ends of the interoperability spectrum: CellProfiler is a standalone tool that would benefit from a loose two-way communication style of integration, whereas VisBio seeks to do many of the same things ImageJ can, but in N dimensions with better separation of concerns, and thus is a natural fit as a suite of ImageJ plugins harnessing the new architecture.

Aims

1. Improve ImageJ's core architecture
 - a) Separate data model from user interface
 - b) Develop extensions framework for algorithms
 - c) Broaden the image data model
2. Expand interoperability with other tools
3. **Grow ImageJ community resources**

See also: imagejdev.org/proposal

29

Lastly, Aim 3 is a fairly broad set of goals intended to foster the idea of community-driven development, including strong web-based resources for both users and programmers. We want to establish a central community resource for accessing software releases, plugins and scripts, source code, documentation, and more.

This aim also specifically targets compatibility with the wealth of existing community-created plugins, macros and other code. This goal will ensure that ImageJ2 is not a reboot, but rather a continuation of ImageJ's development.

For more details on these aims, including the full technical proposal, see the ImageJDev web site.

Outline

- Vision
- Aims
- **Design**
- Progress
- Future Directions

"I haven't failed, I've found 10,000 ways that don't work."

—Thomas Edison

30

Next, we will cover **how** we plan to accomplish our goals. There won't be time to fully explore the specifics, but we will briefly summarize our planned approach.

Design

- Considered several design approaches
 - Iterative (current strategy)
 - Greenfield (new application)
 - Delegation (change IJ1's internals)
 - Adaptation (leave IJ1 alone)
- Adaptation: IJ2 includes IJ1 as a library
- IJ1 and IJ2 grow and evolve together
- More slides during roundtable if interest

31

For the last few months, we have struggled to come up with a design that accomplishes all the goals outlined earlier, while maintaining our guiding principles. We knew we would need to restructure things a bit, and move away from the current ImageJ development model, partly due to the scope of what we want to accomplish, and partly because there is now a larger core development team.

Based on feedback from all involved, we settled on an approach we call Adaptation, where the new ImageJ2 program includes ImageJ1 as a library, communicating with it as necessary to execute plugins and macros faithfully. This approach has seen some success in industry—for example, Adobe overhauled their Flash virtual machine in version 9, but continued bundling the older VM as well for compatibility.

The Adaptation approach will allow ImageJ1 and ImageJ2 to both continue developing, and allow users to gradually migrate to ImageJ2 over time.

Outline

- Vision
- Aims
- Design
- **Progress**
- Future Directions



32

Now we would like to share progress so far across several areas.

All of this work is under heavy development, and it will be several more months before most of it is ready for general use. Our goal is to have a beta of ImageJ2 available some time in the spring.

Progress

- 1) Imglib: an N-dimensional image data model
 - Bio-Formats: reading data
- 2) Automatic plugins for extensible visualization
 - Spectral lifetime image data plugin
- 3) OpenCL-based iterative deconvolution

33

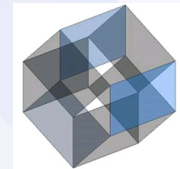
First, we have begun integrating the imglib library with ImageJ, and have a file reader module that uses Bio-Formats to import data as imglib images.

Second, we have a simple prototype of a “display plugin” mechanism, which automatically executes a compatible visualization plugin when an image is first opened, based on the image's dimensional structure—and we wrote a display plugin for visualization of lifetime images.

Third, we did some work translating Bob Dougherty's 3D iterative deconvolution plugin to use OpenCL, which enables GPU-accelerated processing to achieve a six-fold speed increase.

Progress: Imglib

- Added data types backed by Imglib library
 - Currently supports nine pixel types:
 - Signed and unsigned integer, floating point
 - Bit depths: 8, 16, 32, and 64 bit
- Many possible storage strategies
- Type-independent plugins



34

Imglib is a library for N-dimensional image processing in Java, developed at MPI-CBG in Dresden by Stephan Saalfeld and Stephan Preibisch.

We have added limited support for imglib-based image data to ImageJ. This enables ImageJ to work with new pixel types such as 32-bit integer data, but more importantly to take advantage of imglib's flexible container approach. With imglib, your data could be stored in an array in memory—which is how ImageJ currently works—or alternatively could be access your images from a remote database or other source.

Imglib also allows you to write a plugin once that works with all pixel types, rather than needing a special case for each one.

Our current approach for using imglib within ImageJ required changing the ImageJ1 code, but since settling on the Adaptation-based design, we plan to rework this to make it more of an add-on from the ImageJ2 side, rather than altering ImageJ1.

If you want to learn more about Imglib, I encourage you to attend Stephan Preibisch's imglib workshop tomorrow.

Progress: Bio-Formats

- Adapted ImageJ to use Bio-Formats natively for reading file formats
- Files are opened as N-dimensional, imglib-backed images



Bio-Formats is a library developed at LOCI for reading and writing file formats. Fiji currently comes bundled with the Bio-Formats plugins, but we would like to use Bio-Formats as a model for ImageJ's input/output routines.

We wrote a module for loading an imglib image from a file on disk using Bio-Formats. Together with the imglib support, ImageJ2 can natively use Bio-Formats to open image data, preserving the N-dimensional structure.

I'll show this in action in shortly.

Progress: Automatic Plugins

- Plugin author uses Java Annotation to label which dimensions a plugin can handle
- When image is loaded from File/Open IJ checks dimensions and finds matching plugins
- IJ automatically runs unique matching plugin, or displays dialog of choices if several match

```
@Dimensions(required="X,Y,Lifetime", optional="Channel")
public class SLIMPlugInAuto implements IAutoDisplayPlugin {
    ...
}
```

36

Due to the variety of possible in image data these days, we are interested in ImageJ providing context-sensitive visualization, depending on the type of image.

explain example annotation

explain dynamic plugin discovery

We have developed an example plugin for spectral lifetime data, which I'll show now.

Progress: Quick Demo

37

Execute `./ij.sh` in Terminal

File/Open `test_greys.tif`

Show new type on Image/Type menu

Move to later timepoint

Do “make composite” on `test_greys.tif`

File/Open `image.zvi`

Do Plugins/Filters/Floyd-Steinberg on `image.zvi`

Do Edit/Undo to undo it

File/Open the SDT file, explain SLIM Plugin a little

Quit ImageJ

Progress: OpenCL Plugin

- OpenCL: Run software on CPU and GPU cores for fast processing-intensive analysis
- Web services: Invoke code from a remote machine—cross-language, cross-platform
- Methodology for applying iterative speed-up to existing Java code by translating to OpenCL

38

explain the three points

Unfortunately, no demo...

Progress

- 4) ImageJX: pursuing a separation of concerns
- 5) Declarative plugins for greater interoperability
 - CellProfiler connectivity with ImageJ
- 6) Requirements: community feature requests
- 7) Software development methodology and tools

39

A few other areas of progress...

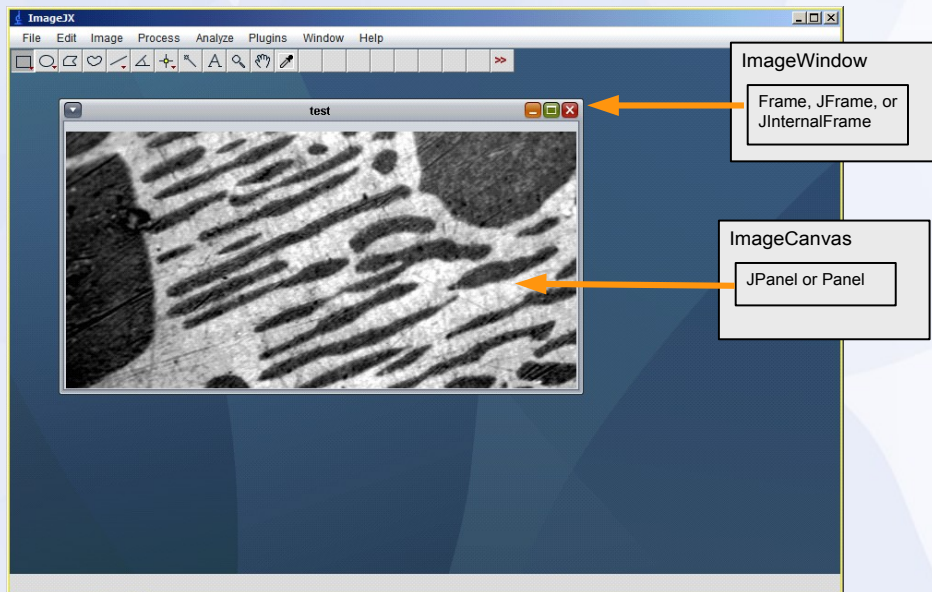
We have been working on a refactored version of ImageJ codenamed “ImageJX” with a more flexible user interface...

I'll cover use of an updated “declarative” plugin mechanism, which has proved useful for integrating other applications such as CellProfiler with ImageJ...

I'll briefly summarize the categories of feedback we received from the ImageJ community regarding what they would like to see in ImageJ 2.0...

And I'll explain some of the tools we've adopted to foster effective development practices.

Progress: ImageJX



40

ImageJX is an attempt to rework the core of ImageJ to have “GUI independence”—a better distinction between the parts of the program that do the actual image processing, and other parts that display user interface on screen.

We have code in development that produced the interface you see here, using a Swing MDI model rather than ImageJ's usual AWT interface. The goal is not necessarily to “port ImageJ to Swing,” but rather to show that an ImageJ user interface could take many forms. This work paves the way for ImageJ in many other contexts, from command line or headless operation mode, to a web-based interface, to use on mobile devices.

Progress: Declarative Plugins

- Existing plugin example:

```
ImagePlus original = WindowManager.getCurrentImage();

GenericDialog gd = new GenericDialog("\tubeness\ Filter");
gd.addNumericField("Sigma: ",
    (calibration==null) ? 1f : minimumSeparation, 4);
gd.addMessage("(The default value for sigma " +
    "is the minimum voxel separation.)");
gd.addCheckbox("Use calibration information", calibration!=null);

gd.showDialog();
if (gd.wasCanceled()) return;

double sigma = gd.getNextNumber();
boolean useCalibration = gd.getNextBoolean();

TubenessProcessor tp = new TubenessProcessor(sigma, useCalibration);
```

41

Johannes Schindelin of the Fiji project came up with a clever way to make plugins simpler, while also allowing them to be run in more contexts. This work goes hand in hand with the ImageJX idea of good separation of concerns—the plugin processing logic should not need to invoke any particular user interface components, but rather merely perform operations on data.

Here is an example to illustrate. *explain*

Progress: Declarative Plugins

- Declarative plugin example:

```
@Parameter(label="Input image")
public ImagePlus original = null;

@Parameter(label="Sigma")
public double sigma = 1.0;

@Parameter(label="Use calibration")
public boolean useCalibration = false;

@Parameter(label="Output image", output=true)
public ImagePlus result = null;

public void run(String ignored) {
    if (original == null)
        original = WindowManager.getCurrentImage();
    TubenessProcessor tp = new TubenessProcessor(sigma, useCalibration);
    ...
}
```

42

continue explanation

It's a “declarative” plugin because it clearly declares input and output parameters, with useful metadata.

No more need to invoke GUI-centric concepts such as `GenericDialog`—less boilerplate code.

Enables greater interoperability. In interactive mode, automatically constructs and displays input dialog. But other modes are also possible.

The annotations standardize the mechanism for declaring plugin inputs and outputs, allowing the plugin to be used by any compatible imaging program, not just ImageJ.

Progress: CellProfiler

- CellProfiler is a tool for executing high-throughput image analysis pipelines
- Achieves better interoperability with ImageJ using the declarative plugin mechanism

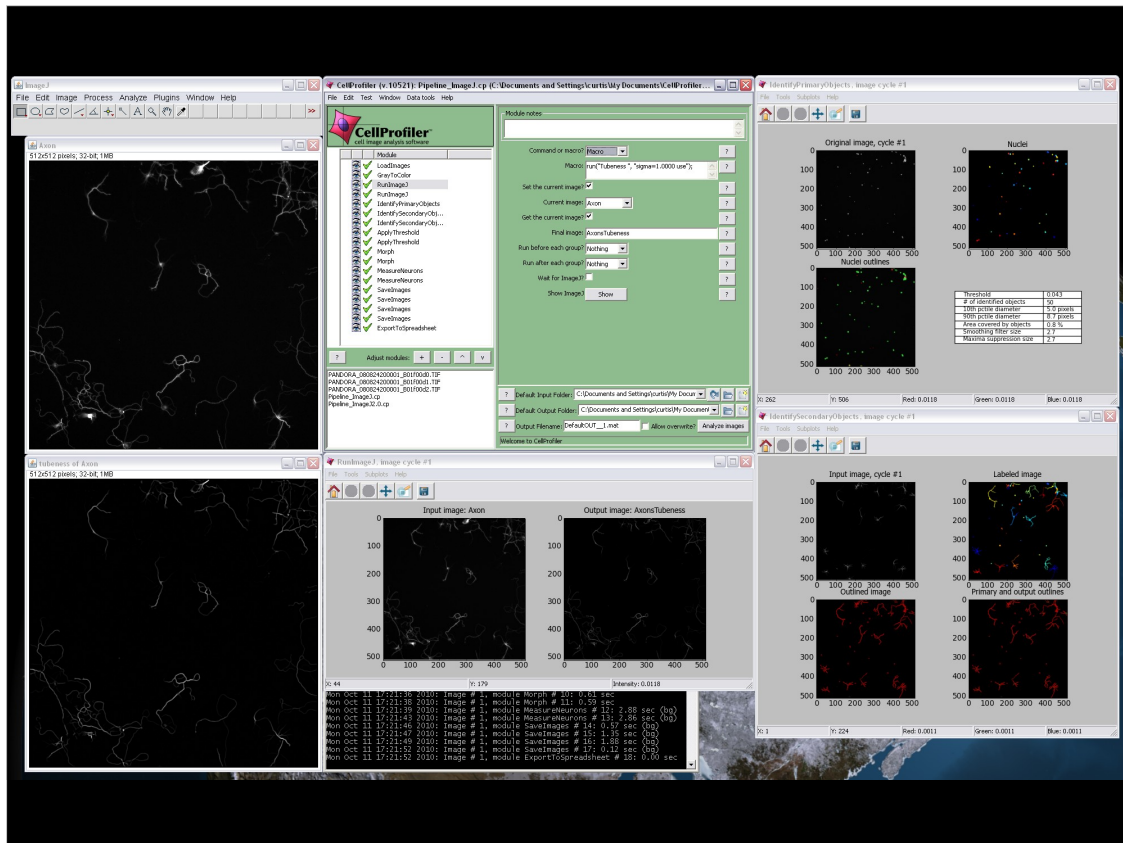
43

CellProfiler is a program developed at the Broad Institute, for performing automated analysis of large numbers of images.

The user defines a pipeline of operations to perform, then repeats that pipeline across many datasets.

CellProfiler recently added the ability to call an ImageJ plugin as part of a pipeline. However, the integration required ImageJ to be displayed onscreen, which is a problem when executing pipelines on a cluster with no user interface.

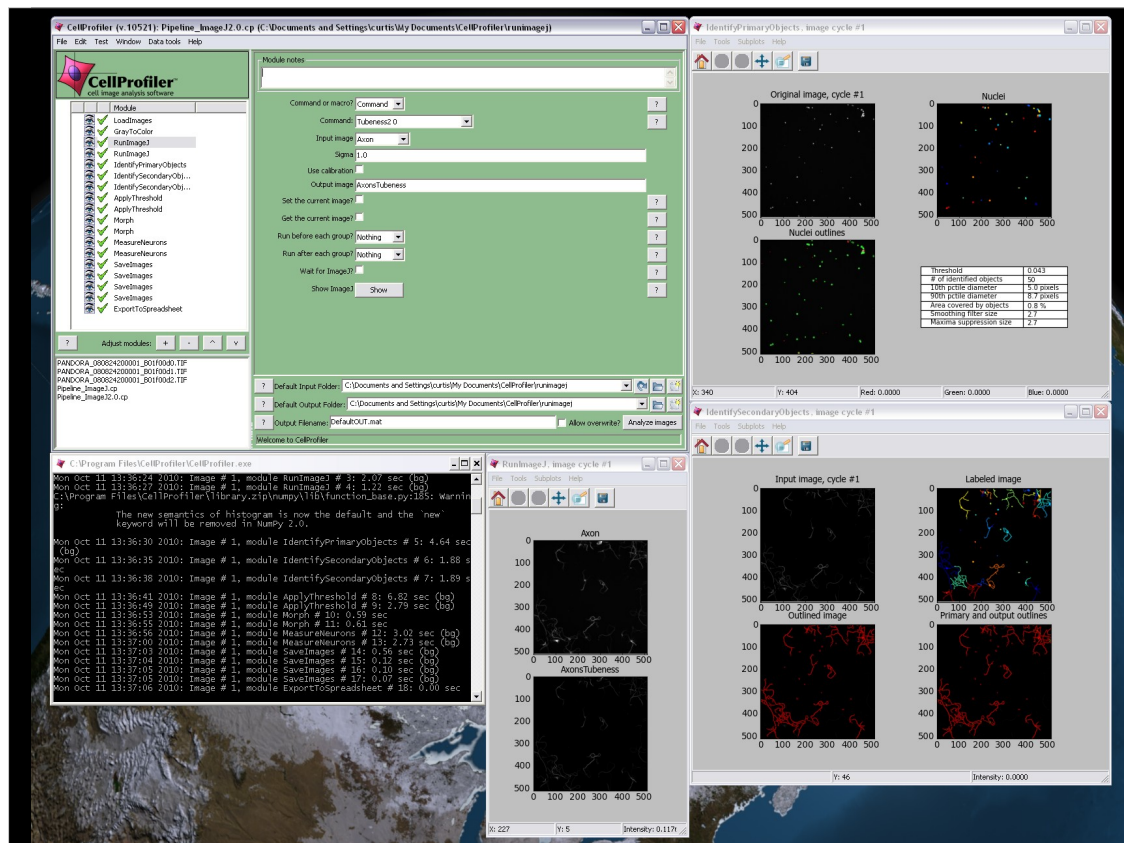
The CellProfiler team was able to improve their support for ImageJ plugins by utilizing the new declarative plugins mechanism.



Here we see CellProfiler interacting with IJ1's standard plugin mechanism. This pipeline calls the Tubeness plugin, written by Mark Longair, Stephan Preibisch and Johannes Schindelin, which filters an image stack to produce a score for how "tube-like" each point in the image is.

Note that the ImageJ windows must be physically shown on-screen, in addition to the CellProfiler interface's windows.

Further, the macro execution shown above is error-prone and can only be accomplished by a user who is very familiar with both applications.



Here we see CellProfiler utilizing the new declarative mechanism. Lee Kametsky translated the Tubeness plugin into a declarative plugin, which enables CellProfiler to more easily integrate with its own user interface. The IJ2 plugin tells CellProfiler the inputs and outputs, as well as helpful text to display next to each field.

Notice that no ImageJ windows need to appear. The interoperability is also more robust; the Broad Institute is already actively using ImageJ plugins with CellProfiler on high-throughput screens on their cluster.

Progress: Requirements

- Gathered feedback from the community
- Major areas of ImageJ
 - Data model & image processing
 - Visualization & user interface
 - Input & output
 - Segmentation & regions of interest
 - Scripting & plugins

46

One of the first things we did was to solicit feedback from community regarding ImageJ 2.0's needed features, and we got a pretty great response. There isn't time to list it all here, but we found that nearly everything mentioned fell into one of the above five categories.

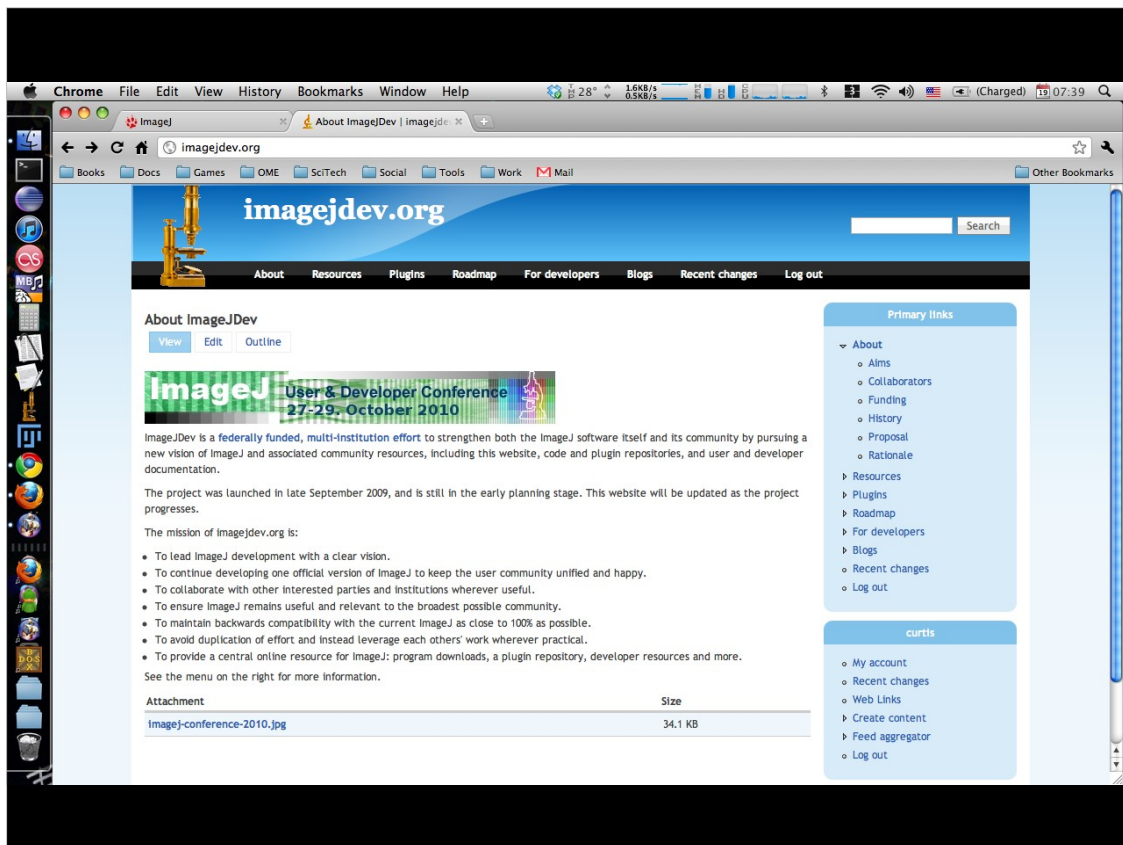
I have quite a few slides detailing individual items, which are available for reference during the round table discussion as needed.

Progress: Development Tools

- Web site
- Unit test suite
- Continuous integration: Hudson
- Source control: Subversion & Git
- Project management: Maven & Trac

47

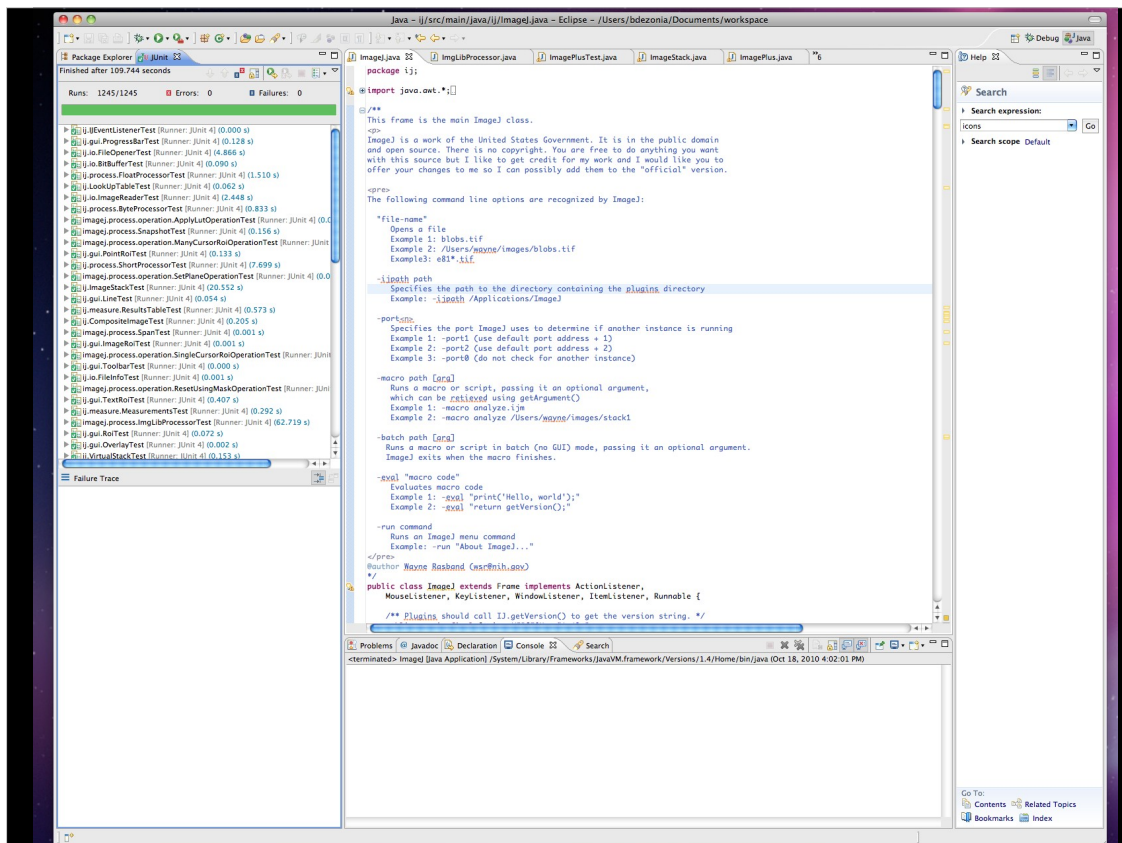
As I mentioned earlier, we want to have an effective development process. We are using several standard tools and methods to assist with this goal.



This is our website for the ImageJDev project, with documentation and links to the other tools.

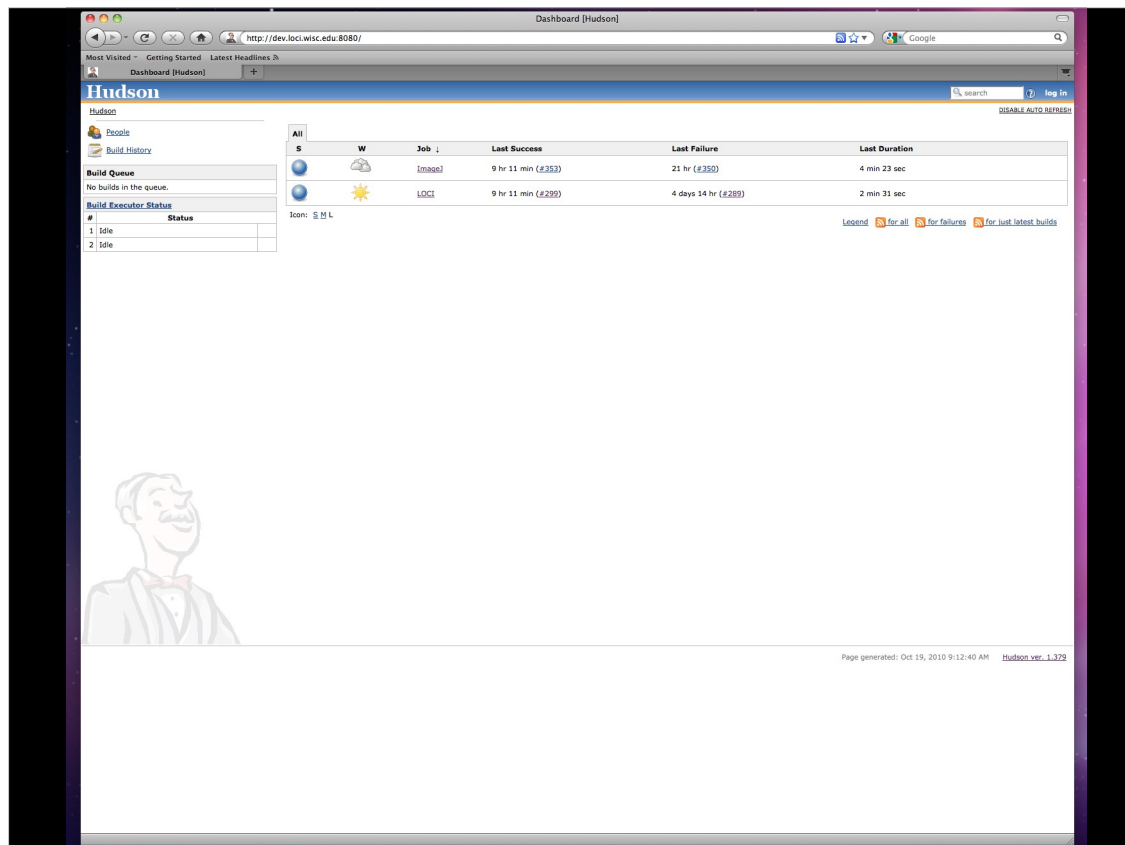
It is set up using the Drupal content management system, so that we can easily extend the functionality, and so that multiple users can collaboratively edit the site.

We still have a lot more work to do getting more content on the site, but it is a start.



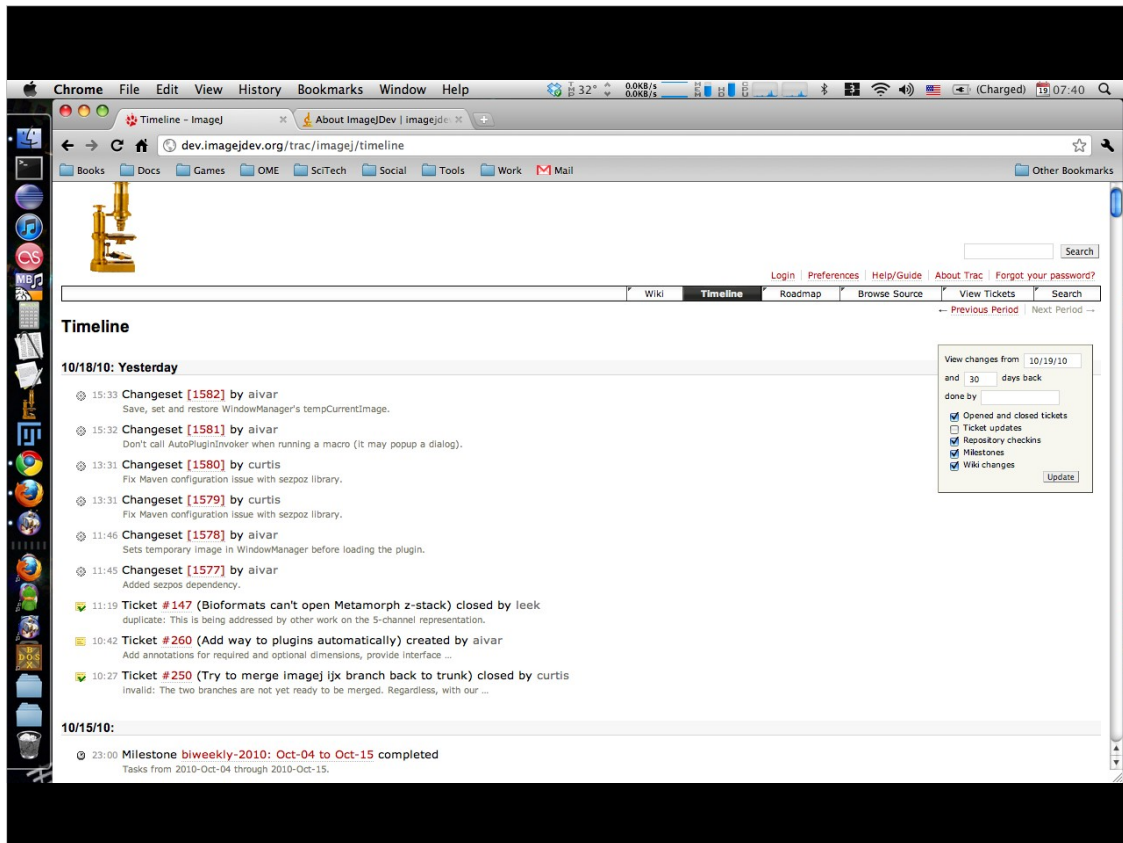
We wrote some unit tests to ensure that existing ImageJ behavior is preserved when code is changed. We currently have test cases for around 50 core ImageJ classes, though more are still needed for full coverage.

Here we see the Eclipse development environment executing our many unit tests: 1,000 robot monkeys each repeating a different little task. The green checkmarks mean the tests are passing.



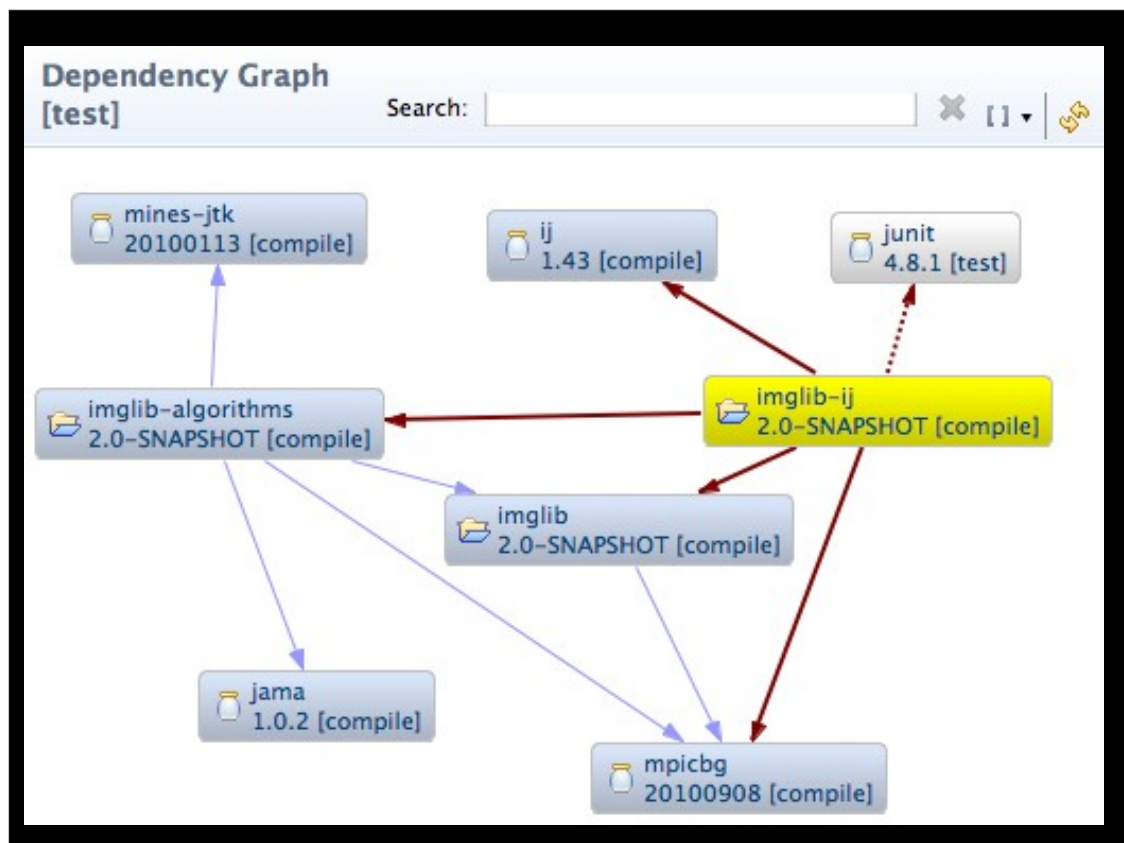
Hudson, our continuous integration system, makes it less likely for us to break the program without noticing for a long period of time by automatically performing builds, running tests, and emailing us if something goes wrong.

Here, Hudson reports that all is well with the latest code—though the little cloud next to ImageJ means that there was a failure one of the last five times. If the build or tests are broken, the blue circle turns red, and the weather gets stormy.



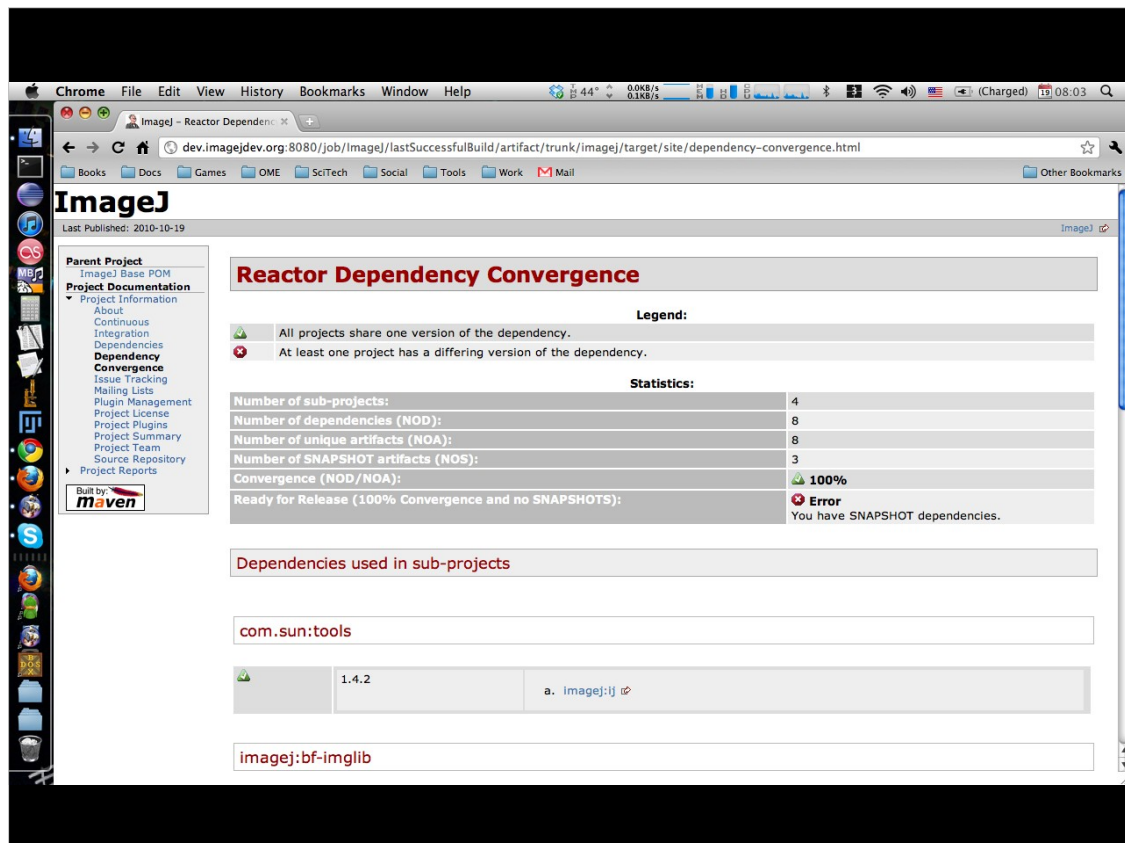
The web-based Trac project management system makes it easier to organize everyone's pending tasks, and keep track of the problems people have reported with ImageJ.

This is a view of the Trac showing a history of activity over the past 30 days. A Changeset means that somebody made a change to the code, while a Ticket event indicates progress or discussion on a bug or task in the bug tracking system.



The Maven project management and build tool helps to keep dependencies organized. While I have touted modularity as a good thing, as you develop more and more modules that depend on one another, it helps to have a way to visualize these relationships. Here we see a graph of project dependencies, generated in Eclipse using the Maven plugin, for imglib, which currently consists of four modules.

describe briefly



One great feature of Maven is the ability to generate a website for your project with various reports and code analysis. Here we see a Maven-generated site report that summarizes the dependencies of our ImageJ 2.0 development code.

Maven integrates very nicely with a large number of powerful project management tools, such as automated bug detection and code coverage analysis. And as new tools are developed, it's likely that the software development community will integrate them with Maven as well, making it easier for us to take advantage of them.

Outline

- Vision
- Aims
- Design
- Progress
- **Future Directions**

“Your task is not to foresee the future, but to enable it.”
—Antoine de Saint Exupéry

54

Hopefully our vision and aims gave you a pretty good idea of our future directions for the project. But we'll briefly summarize what to expect in the next year and beyond.

Future Directions

- Pursue Adaptation design for IJ 2.0
- N-dimensional image data model
- Investigate standards useful to ImageJ
 - Rich client platform for user interface
 - Modularity and interoperability: e.g., OSGi
 - ROIs: e.g., JHotDraw
- Improve headless behavior
- Implement community requirements

55

Our primary goal is to create a version of ImageJ 2 that can communicate with the current ImageJ 1 application, operating as faithfully as possible to preserve backwards compatibility and keep existing plugins and macros working.

For the 2.0 version, we will pursue an interface-driven, modular design with good separation of concerns, based on the current ImageJ 1 codebase, but without being overly constrained by it. In general, if we need to change something, the Adaptation design gives us the ability to do so without breaking backward compatibility.

We will do our best to apply industry standard software engineering tools and practices, so that ImageJ development can be driven not just by the ImageJDev project, but by the community as a whole.

Summary

- What Will ImageJ 2.0 Do for Me?
 - Work with existing plugins and macros
 - Work with new, exciting plugins and scripts
 - Handle larger, more complex datasets
 - Multidimensional visualization tools
 - Easier to link with other software
 - Easier plugin management

56

To conclude, ImageJ 2 will work with existing plugins and macros, while enabling the use of new kinds of plugins and scripts as well. In particular, it will support images that are larger and more complex, with additional dimensions that can be visualized in a variety of ways. For developers, it will be easier to invoke ImageJ from other software—and for users it will be easier to manage which plugins you have installed, and keep them up to date.

Acknowledgements

- **Principal Investigators**

- Kevin Eliceiri (LOCI), Rudolf Oldenbourg (MBL), Anne Carpenter (Broad)

- **Developers**

- Grant Harris, Barry DeZonia, Aivar Grislis, Rick Lentz (ImageJDev)
- Lee Kamentsky, Adam Fraser (CellProfiler)

- **Collaborators**

- Wayne Rasband (ImageJ)
- Pavel Tomancak, Johannes Schindelin, Albert Cardona (Fiji)
- Stephan Preibisch, Stephan Saalfeld (Imglib, Fiji)
- Mark Longair, Jean-Yves Tinevez (Fiji)
- Jason Swedlow, OMERO development team (OME)

57

I would like to thank everyone involved in the project, including our collaborators, Mark Longair for his tubeness plugin, and Jean-Yves Tinevez

Also thank the other Fiji developers, including Mark Longair (tubeness) and Jean-Yves Tinevez (imglib)

Also thank NIH and the stimulus funds

Discussion

- Comments? Questions?
- Thoughts on what ImageJ 2.0 should be?
- Ideas from the community

Design Approaches

Design Approaches

1. Iterative

- Pro: No project forks
- Pro: Maintains compatibility whenever possible
- Pro: Brings code “under test”
- Con: Heavily constrained by the existing design
- Con: Development is slow

2. Greenfield

- Pro: Great flexibility
- Pro: Rapid development
- Pro: New code is “under test”
- Con: No compatibility
- Con: Forks the project
- Con: Loses legacy codebase's “embedded knowledge”

60

Let's start by talking about a purely evolutionary approach to software development. As features are needed, they are added to ImageJ one by one. Eventually, when we meet our goals, we declare a “2.0” release of ImageJ, and continue from there.

This approach is how ImageJ has been developed for the past decade, and it has a lot going for it—in particular, with care it is possible to maintain compatibility with existing plugins indefinitely.

However, the compatibility comes at an increasingly steep cost. As ideas are developed and improved, the prior paradigms must be kept in place, and the code becomes increasingly hard to understand. Worse, some paradigms end up being very difficult to shoehorn into the existing code structure.

Design Approaches

1. Iterative

- Pro: No project forks
- Pro: Maintains compatibility whenever possible
- Pro: Brings code “under test”
- Con: Heavily constrained by the existing design
- Con: Development is slow

2. Greenfield

- Pro: Great flexibility
- Pro: Rapid development
- Pro: New code is “under test”
- Con: No compatibility
- Con: Forks the project
- Con: Loses legacy codebase's “embedded knowledge”

61

Contrast that with a naïve “greenfield” design, where we redesign the software from the ground up. In a very real sense, the new application is not a version upgrade, but rather a brand new program.

This approach is very common when developers feel they are hitting the ceiling on what is feasible with the existing “legacy” software. Of particular advantage is the fact that there are very few constraints on the new design.

However, a greenfield design inherently has zero compatibility with the legacy application—existing code will not work with the new application unless it is reworked to use the updated paradigms.

Lastly, while the new design may apply conceptual lessons learned from the legacy application, it loses the “embedded knowledge” present in the existing codebase, discovered through years of effort, blood, sweat and tears.

Design Approaches

1. Iterative + 2. Greenfield

?

62

Both approaches have advantages, but also serious difficulties—is there a combined approach that achieves the best aspects of both?

Design Approaches

Approach #3: Delegation

- Good compatibility
- Good design flexibility
- But very disruptive of legacy work

63

One possibility we seriously considered is a delegation model. With this approach, we create a new IJ2 application, and then transform IJ1 over time to rely on IJ2 routines for its core functionality.

In many ways this scheme seems promising, because it keeps compatibility in mind, while allowing substantial freedom in the new design. In some cases, the IJ1 application could even gain access to new IJ2 features “for free.”

Unfortunately, in some ways the new design is still inherently constrained by the needs of the legacy application. That is, IJ2 must be capable of doing everything IJ1 can do, in a compatible paradigm.

Lastly, as more and more IJ1 code is refactored into delegation calls to IJ2, it becomes increasingly necessary to understand the IJ2 design as well, making continued IJ1 development difficult.

Design Approaches

Approach #4: Adaptation

- Nearly perfect compatibility
- Smooth transition from legacy code
 - Legacy work continues as long as needed

64

These issues lead us to propose a different design based on adaptation. The trick is to create a compatibility layer, or “adapter,” that converts data between the IJ1 and IJ2 data representations, transforming our problem of compatibility into one of interoperability.

In some ways this approach is the inverse of the delegation model: instead of forcing IJ1 to depend on IJ2, it's the other way around. Specifically, we enable IJ2 to use IJ1 as a library to execute existing plugins, transforming the data between representations as needed. (In many cases the transformation will be very efficient, as image data structures can share references to primitive arrays.)

Another major advantage of this approach is that IJ1 development can continue until IJ2 has reached full maturity. During the transition, users needing maximum stability can continue using IJ1, while those desiring new features can adopt IJ2.

Design Approaches

Approach #4: Adaptation

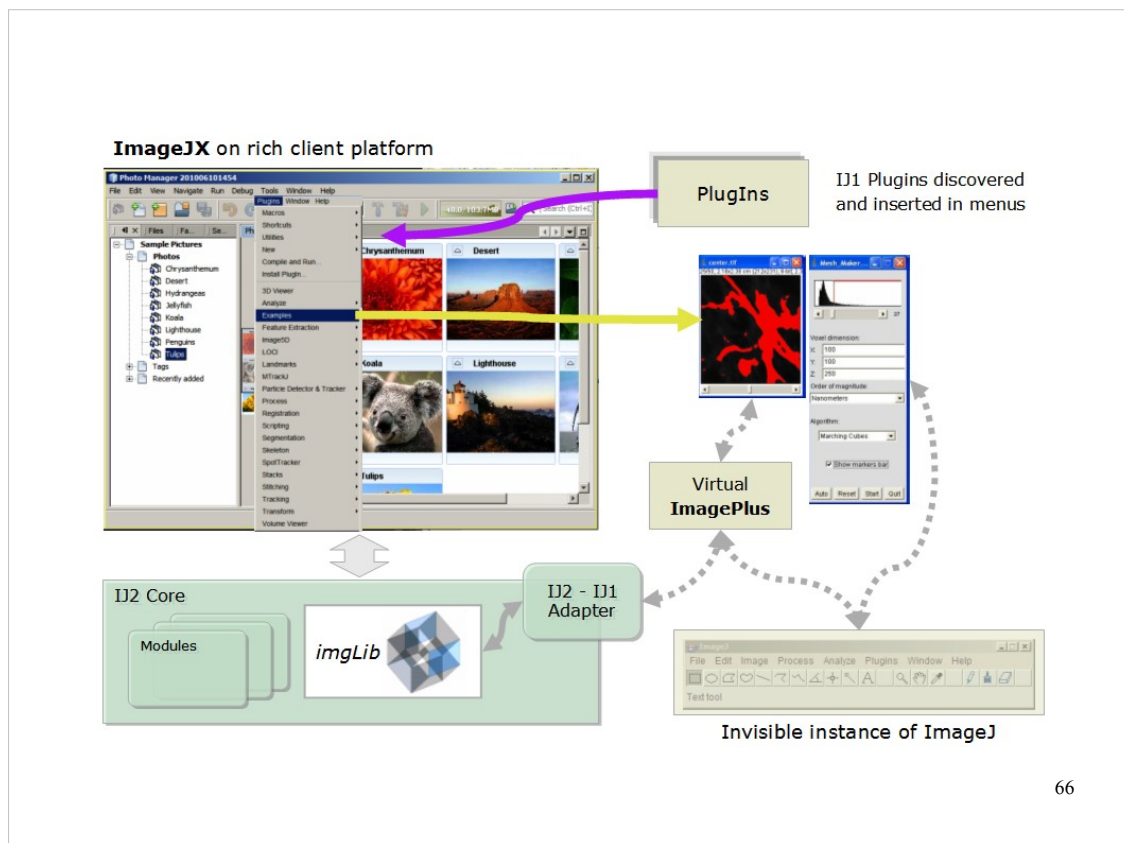
- Some limits to interoperability
- Harnesses “embedded knowledge” of legacy work without being constrained by it

65

It is worth pointing out that this design does have some minor interoperability limitations. Specifically, new IJ2 data structures may not translate perfectly to the old IJ1 data model. For example, if IJ2 supports a new kind of ROI, or a new pixel type, it might not be expressible in terms IJ1 can understand.

Fortunately, in such situations, there are unlikely to be existing IJ1 plugins that would benefit greatly from the new structures—and if there are, they can be updated to run natively in IJ2.

Lastly, we can continue to benefit from the last decade of effort by branching the ImageJ 2.0 codebase from IJ1, rather than starting from scratch with a purely greenfield design. Because IJ2 is not directly responsible for compatibility with IJ1, we are free to change the design as needed to encompass new features and ideas.



66

This diagram illustrates how the adaptation design would allow IJ2 to interoperate with IJ1 as a library.

Existing IJ1 plugins are discovered and listed in the IJ2 application's menu, as normal. When one of them is invoked, the input data is transformed via the adapter into an IJ1-based representation such as an ImagePlus object. If an IJ2 plugin is later invoked on the result, it is transformed back into an IJ2-based representation such as an imglib image object.

Although this description is a simplification of the procedure needed, hopefully it illustrates the essential principle.

Community Use Cases

Use Cases: VisBio

- Limited support for large datasets
 - Image planes larger than 2GB
 - Datasets larger than available RAM
 - VirtualStacks cache only one plane at a time
- No support for 3D visualization
 - Volume rendering
 - Arbitrary slicing
 - Realtime animation
- Also needs better support for ROIs

Use Cases: Slim Plotter

- No support for new dimensions
 - Emission spectra
 - Lifetime
 - Polarization
- No support for processing inherent to viz
 - Exponential curve fitting
 - Spectral unmixing

Use Cases: Fiji

- Distributing plugins is external to ImageJ
- Keeping everything up to date is complex
- No standard for documenting plugins
- Not easy enough to prototype algorithms
 - Plugins require too much boilerplate code
 - No modular command framework for using Macro Recorder with scripts
 - Case logic for multiple pixel types is messy
- AWT dependencies preclude headless use

Use Cases: TrakEM2

- No support for displaying registered images
 - No display mechanism for multiple image tiles
 - No mechanism for transformation from data to display (e.g., affine)
- Regions of interest are limited
 - No vector-based ROIs (i.e., ROIs are bitmasks)
 - Multiple ROIs are tacked on (ROI Manager)
 - Confusing interplay between ROIs, masks & thresholds with measurement tools

Use Cases: ROIs (Michael Doube)

- Recently I've been frustrated by ROI's being limited to 2D. With the emerging utility of the 3D viewer and the proposal that ImageJ 2.0 handles N-dimensional data, it makes sense that ROIs should keep up with this development.
- In other words, in an N-dimensional image, one should be able to specify and visualise an N-dimensional ROI. So you can have a 3D VOI, and a 4D VOI with time limits (or even changing shape over time), or limit the ROI to a channel (5D).

Use Cases: ROIs (J-Y Tinevez)

- I recently tried to code weird shapes as ROIs in ImageJ. They were the results of a segmentation with constrained shapes. Because I wanted to have something nice for the user, The ROIs had to be mouse-interactive (resizable, moveable etc..). I had a difficult time.
- Johannes proposed on the Fiji-devel list an abstract class whose goal was to facilitate this interaction.
- But we still gave to comply to ImageJ `ij.gui.Roi` master class, which is a concrete class in charge of drawing rectangle ROIs. Inside this class, there is everything: the logic to draw it, to interact with the user, with the image container, and the image data. Any homemade ROI must inherit from this class, there is no interface to implement.

Use Cases: ROIs (J-Y Tinevez)

- What I would like to propose here is to go for an interface hierarchy for ROIs, that is well decoupled, and that would allow the flexible design of new ROIs.
- We use ROIs for many purposes, for instance:
 - user interaction
 - draw a rectangle to crop an image
 - measure intensity with a complex area
 - add non-destructive annotations
 - as input/output for plugins, for instance a result of segmentation
- From this you can see that they need to:
 - know how to draw themselves as an overlay
 - comply to some interface to be an input of some plugins
 - know how to interact with mouse clicks and drag

Use Cases: μ Manager (N. Stuurman)

- 1. The Brightness/Contrast tool. Display of the histogram cannot be reliably set to the dynamic range of the camera (i.e., it always automatically goes back to the range of the minimum and maximum pixel value in the image, which can be extremely deceptive). No gamma correction. No method to update histogram when the image changes. No log display of the histogram. We ended up writing our own, but things are still clunky because acquired images (shown in a modified Image5D viewer) can only be controlled by the ImageJ B&C tool.

Use Cases: μ Manager (N. Stuurman)

- 2. Lack of plugin API. We have been bitten a number of times by internal changes in ImageJ breaking our code. Wayne is very responsive, but this still causes confusion.
- 3. Lack of standard for Multi-Dimensional viewer. We ended up using Image5D viewer, Hyperstacks came later. My impression is that the UI of Image5D is easier for users than the UI of Hyperstacks. In any case, we will be helped by a standard viewer for multi-dimensional images that integrates nicely with other ImageJ tools (like 3D viewers), and that is extensible (we do need to add a number of buttons that interface with image acquisition).

Use Cases: μ Manager (N. Stuurman)

- 4. MDI versus SDI. Not sure if this was on your list already (all of you have certainly debated this in the past!), but it seems that many people prefer the MDI model. On the Mac, it is pretty weird that a single application has different menus depending on which window you select (in our case, ImageJ windows versus Micro-manager window).

Use Cases: Miscellaneous

- G. Landini: no color space support (e.g., HSB)
- F. Hessman: domain coordinate systems
 - S&S are planning support within imglib
 - ImageJX consensus is to punt on this for now
 - Need to find a group with this use case first
- Legacy AWT interface limits use of Swing
 - ImageJ cannot use different L&Fs
 - AWT is missing features (JSpinner, JInternalPane)
 - Swing development is active, unlike legacy AWT

Use Cases: Compatibility

- Advantage of ImageJ: wealth of existing code
- Problem: ImageJ2 will break that code
- Examples:
 - `ImageProcessor.getPixels()`
 - All non-private, non-final fields
 - Subclasses created to sidestep API issues
 - Even private fields—`setAccessible(true)`

Use Cases: Interoperability

- FARSIGHT: ITK-driven segmentation routines are difficult to use from Java
- CellProfiler: How can scientists combine workflows between CellProfiler and ImageJ?
- OMERO: Database-backed images are kludgy
- Others: KNIME, Endrov, BioImageXD, PSLID...

Use Cases: Performance

- Traditional tradeoff between space & time
- Tradeoff between generality & performance
 - Moving toward generality requires that we remain aware of performance issues
 - But flexibility and usability remain paramount
- OpenCL is promising but negates many of imglib's gains in generality

Components of ImageJ2

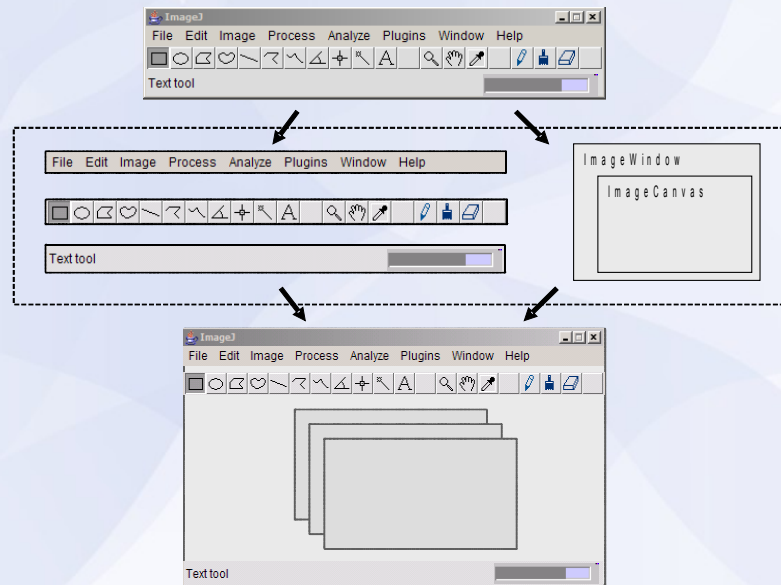
- Relevant technologies
 - 1)**Data model** – imglib library
 - 2)**Display** – Java AWT, JAI, Swing, RCP
 - 3)**Input/output** – Bio-Formats architecture
 - 4)**Regions of interest** – Java AWT, JHotDraw, OME-XML
 - 5)**Scripting & plugins** – Java 6 Scripting Framework
- More exploration of some technologies needed

ImageJX: Separation of Concerns

Decouple GUI dependencies

- Alternative GUI configurations (e.g., Swing SDI/MDI)
- Headless operation
- Incorporation into application framework
- Easing use as a library

GUI Decoupling



85

Image Processing GUI components with default implementations in javax.swing

Mostly Jpanels

Other developers can provide alternate implementations of the interfaces we define.

Dynamic Plugin Discovery

- Declarative Registration using Annotations
 - Menus, etc., are built dynamically from plugin declarations
- Classes do not need to be loaded
 - Uses 'compile-time caching' (SezPoz)
- 'Automatic Plugins'
 - I/O (Bio-Formats reader)
 - Display—invoke a plugin in response to a particular kind of data being opened

Dynamic Plugin Discovery

```
package demo.plugin1;

import demo.api.MenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

@MenuItem(
    label = "Exit",
    menu = "File",
    icon = "demo/plugin1/movieNew24.gif",
    bundle = "demo.plugin1.properties")

public class ExitAction implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

Toward Modularity & Extensibility

- Use interfaces, abstract classes, factories
 - Replaceable implementations
 - Enables dynamic assembly
- @ServiceProvider (e.g. SavePrefs)
- CentralLookup
- 'Injectable Singletons'
- EventBus
- Context / Selection management