
From Lab to Launch

**Building Instrumentation Software that
Everyone Will Love**

Contact

info@accendero.com

<http://www.accendero.com>



Table of Contents

From Lab to Launch

Table of Contents	1
Executive Summary	2
Introduction: The Hidden Software Challenge in Scientific Instrumentation	4
Case Study 1: Web-Based Architecture for High-Throughput Microscopy	7
Case Study 2: Desktop Architecture for Genetic Analysis	12
Conclusion: Your Path to Instrumentation Software Success	17
About Accendero Software	19

1/

Executive Summary

Scientific instruments generate a deluge of data. Powerful as they are, these instruments often fail to play nice with other tech in the lab, making the software which powers and integrates them a potential bottleneck.

Software needs to handle data complexity and be user-friendly enough for diverse lab teams.

Case 1: Web-Based High-Throughput Microscopy

A commercial microscopy company needed to incorporate custom machine learning algorithms and multi-user collaboration in order to meet customer expectations. Their web-based solution effectively delivered:

- Full python machine learning algorithm integration
- Seamless collaboration on cloud-scale image sets
- Flexible deployment (either cloud or on-premises)

Case 2: Desktop Genetic Analysis Application

A research laboratory needed its sophisticated benchtop genetic instrumentation that was designed for complex IT environments, to work on location. Meaning it had to run on both the lab desktop and on their personal laptops. Their Electron-based solution achieved:

- Installation on any desktop without the need of IT support
- Air-gapped operation capability
- On-budget development cost

Key Insights

Know your constraints: Algorithm complexity and IT environment should determine your software architecture

Prototype early: Validate performance before full development

Match deployment to reality: Every solution cannot solve every problem

2/

Introduction: The Hidden Software Challenge in Scientific Instrumentation



Modern scientific instruments are marvels of engineering—sophisticated hardware capable of generating data at unprecedented scale. Increasingly, the difference between labs that gain full advantage of their instruments isn't the hardware itself, but the software that drives them; software that cooperates with other lab systems and turns raw data signal into scientific insight.

Software is often required to meet competing objectives:

Powerful analysis

- Sufficiently robust to manage complex algorithms, machine learning models, image processing pipelines, and statistical analyses—pushing computational limits.
- Sufficiently fast to deliver real-time immediate feedback, interactive visualization, and responsive interfaces.
- Sufficiently collaborative to be used by multiple users analyzing shared datasets in multiple locations.

Portability and IT-friendly

- Functional independence from IT, which often has its hands full or has limited bandwidth.
- Users running Windows, MacOS, Linux desktops in the same lab

- Security concerns such as strict firewalls, air-gapped facilities, and regulatory compliance.
- Not every lab can afford enterprise infrastructure.

Striking the right balance between powerful analysis and user friendliness can be tricky.

Two Potential Solutions for Success

Success requires choosing the proper architecture to satisfy your specific constraints:

Web-Based Architecture—When Algorithms Drive Requirements

- Your algorithms require specific languages (Python for machine learning, R for statistics)
- Users need to collaborate on shared datasets
- You can ensure reliable network connectivity
- IT support is available for deployment

Pros: Enables algorithmic sophistication

Cons: Requires more IT infrastructure

Desktop Architecture—When Deployment Drives Requirements

- Algorithms can be reimplemented to be portable
- IT support is minimal or hostile to new systems
- Air-gapped operation is required
- Portability is paramount—the software must work anywhere the scientists work

Pros: Simpler deployment

Cons: Limited algorithmic complexity

The Path to Successful Launch

This whitepaper presents two case studies demonstrating how successful organizations navigated these trade-offs:

1. A commercial microscopy company that chose web-based architecture to deliver powerful machine learning capabilities
2. A research laboratory that chose desktop architecture to ensure broad compatibility

Both succeeded because they matched their architecture to their reality—not to an idealized vision of how laboratories should work.

3/

Case Study 1: Web-Based Architecture for High-Throughput Microscopy

The Challenge

A biotechnology company was preparing to launch an innovative high-throughput microscopy system. The hardware was impressive, but their real innovation was in their proprietary machine learning annotation algorithms.

Their challenge: deliver software that would harness these sophisticated algorithms while providing the responsive, interactive experience their customers expected from modern instrumentation.

Specific Requirements

Data volume: Each imaging run generated gigabytes of high-resolution images

Algorithm complexity: Python-based machine learning models

User interaction: Scientists needed to zoom, pan and manipulate annotations

Collaboration: Multiple researchers analyzing the same datasets simultaneously

Deployment flexibility: Certain customers required cloud infrastructure; others required on-premises installation

The Constraints

These factors shaped their architectural decision:

Algorithm investment: Significant previous and on-going development in Python—rewriting their code was not feasible

Customer profile: Their marketing targeted academic labs and biopharmaceutical companies with IT permissions for cloud

The Solution

We architected a web-based application using Django REST + React. Users could operate through a browser, yet the team preserved the key advantages of their sophisticated algorithm.

This project was completed in 11 months for the initial release.

Architecture Highlights

Django REST backend: Native Python environment for machine learning algorithms

React + Canvas frontend: Hardware-accelerated rendering for smooth image manipulation

Docker containerization: Identical on-premise and cloud deployment

Implementation: From Prototype to Production

Stage 1: Performance Validation

Challenge: Could their web technology deliver the responsiveness that users expected?

Approach:

- Built a minimal Flask + vanilla JavaScript prototype
- Implemented critical features: image loading, zooming, and panning
- Tested with actual datasets

Result: User acceptable response times proved this web architecture was viable

Timeline: 4 weeks

Stage 2: Production Development

Challenge: Transform the prototype into commercial-grade software

Key developments:

- Flask → Django for authentication, permissions, and data management
- Vanilla JS → React for maintainable, testable code
- Image sharing and annotation collaboration
- Seamless connection to their existing Python machine learning pipeline

Result: A feature-complete application meeting all requirements

Timeline: 6 months

Stage 3: Commercial Release

Challenge: Ensure successful deployment across diverse customer environments

Launch preparation:

- Deployment packages using Docker compose for local development and AWS ECS for cloud
- IT deployment guides and internal documentation
- Testing on customer infrastructure during early access
- Post launch support featuring performance monitoring and update pipelines

Result: A successful launch that met all critical requirements

Timeline: 4 months

Key Success Factors

Early performance validation: The four week prototype prevented later implementation failure

Preserve existing investment: Keeping Python algorithms accelerated their time-to-market

Flexible deployment: Identical Docker containers for on-premises and cloud

Progressive enhancement: We started with core features and added advanced capabilities iteratively

User-centric design: Extensive user testing ensured their software met real-world needs

Lessons Learned

Successful recommendations:

- Django's maturity provided a robust foundation for commercial software
- React's ecosystem accelerated frontend development
- Docker deployment simplified installation across diverse environments

Considerations for the future:

- Potential customers could struggle to adopt a sophisticated cloud or on-premises system
- Using Django on systems with massive data scale can reach a limit. A future transition to a federated backend may be necessary.

The Bottom Line

By choosing web-based architecture, this company successfully launched instrumentation software that differentiated their hardware in a competitive market. The ability to integrate sophisticated Python-based machine learning algorithms directly into their platform created a competitive product that a simpler desktop application couldn't match.

Most importantly: They met their launch deadline and budget, all while delivering software that scientists could efficiently use.

4/

Case Study 2: Desktop Architecture for Genetic Analysis

The Challenge

A research laboratory had acquired a cutting-edge benchtop genetic analyzer but faced a software problem. The instrument's default software couldn't accommodate their novel analytical methods—proprietary algorithms they'd developed over their years of research. They needed custom software that would integrate their methods and work reliably in their complex IT environment.

Specific Requirements

Algorithm integration: Custom statistical methods

Deployment simplicity: Installable without IT support on various personal computers

Air-gapped operation: Many experiments performed in a secure facility without internet

Budget reality: A limited total budget, including development and deployment

The Constraints

Their laboratory environment presented unique challenges:

IT complexity: The station where the instrument was located was air-gapped

Hardware diversity: Researchers used personal laptops (Windows, Mac, some Linux)

User expertise: Scientists who "just wanted it to work" without significant technical setup

Maintenance reality: No dedicated IT support after deployment

The Solution

We architected a desktop application using Electron that could run anywhere while incorporating their custom algorithms and meeting their budget constraints.

Architecture Highlights

Electron framework: A single codebase for Windows, Mac, and Linux

React frontend: Familiar web technologies for rapid development

Node.js backend: JavaScript runtime with file system access

Algorithm port: Reimplemented statistical methods from Python to JavaScript

Local SQLite database: No server required and all data would stay on the user's machine

Embedded everything: The entire software system was packaged in a single installer

Implementation: From Concept to Deployment

Stage 1: Algorithm Feasibility

Challenge: Designing a methodology for proving our hypothesis that JavaScript would meet their requirements

Approach:

- Analyzed algorithm complexity
- Prototyped core methods using JavaScript libraries

- Validated our results against their initial implementation with test datasets loaded in the browser

Result: Our JavaScript implementation matched the Python output

Timeline: 2 weeks

Stage 2: Rapid Development

Challenge: Deliver a complex scope that would meet the important requirements of all stakeholders while also staying within a strict budget

Strategic decisions:

- Built as a web app first for rapid iteration
- Researchers could test a web version via browser without installation
- Feature prioritization that focused on the core workflow and deferred any nice-to-haves
- A clean, simple interface that minimized training

Key features implemented:

- File import
- Custom algorithm processing pipeline
- Interactive result visualization
- PDF report generation
- Optional user profiles for shared computers

Result: A fully functional web application validated by user testing

Timeline: 8 weeks

Stage 3: Desktop Packaging

Challenge: Convert the web application into standalone desktop software

Technical approach:

- Embedded React app with Node.js backend in ElectronJS
- Installer for easy installation on each platform

Result: A fully functional desktop application on all target systems

Timeline: 3 weeks

The Results

Installation success: 100% successful installation without IT support

Platform coverage: Running on 5 different computer configurations

Budget achievement: Delivered at budget

Timeline: 18 weeks from the kickoff to the deployment

Key Success Factors

Algorithm simplicity: Their lab technology made JavaScript algorithms viable

Web-first approach: Browser testing during development reduced revision cycles

User involvement: Regular feedback prevented expensive late-stage changes

Electron maturity: This framework handled cross-platform complexity without significant implementation

Realistic scope: Prioritized essential features to meet their budget

Lessons Learned

What Worked

- JavaScript's performance was sufficient

- Electron deployment was successful to targeted systems
- Simple architecture ensured simple maintenance

Trade-offs accepted:

- Kept analysis simple as complex machine learning would require a different approach
- Single user focus with no real-time collaboration features
- No cloud backup or sharing
- Users must download new versions and update manually

The Bottom Line

By choosing desktop architecture, this research laboratory got exactly what they needed: software that worked reliably in their environment, without IT infrastructure or ongoing support. Our simplified approach met their budget and delivered a solution that scientists could install and use immediately.

Most importantly: This software “just works” with minimal maintenance, enabling their research rather than hindering it.

5/

Conclusion: Your Path to Instrumentation Software Success

Architecture Determines Destiny

The microscopy company chose **web-based architecture** because their sophisticated machine learning algorithms and collaborative requirements justified the IT complexity. They launched on time with differentiated capabilities that command the market.

The research laboratory chose **desktop architecture** because simplicity and portability mattered more than advanced features. They stayed on budget while delivering software that works anywhere their scientists work.

Both succeeded because they matched their architecture to their requirements—not the other way around.

Your Decision Framework

Choose Web-Based Architecture When:

- Algorithms require specific languages (Python, R, MATLAB)
- Users need real-time collaboration
- Data volumes exceed local storage
- IT infrastructure exists or can be required
- Ongoing updates and cloud features add value

Choose Desktop Architecture When:

- Deployment simplicity is paramount
- IT support is minimal or restrictive

- Air-gapped operation is required
- Algorithms can run in portable languages like JavaScript
- Budget constraints are tight

The Path Forward

Start with an honest consideration of your constraints:

Assess your algorithms: Can they be reimplemented or must they stay in the current language?

Understand your users: What's their IT reality?

Define your success: On-time launch? Under budget? Feature differentiation?

Prototype early: Validate needed performance before full implementation

Plan deployment: How will your software actually reach users?

Common Pitfalls to Avoid

Underestimating deployment: Installation challenges will kill adoption

Ignoring IT reality: The best software is useless if your IT won't approve it

Forgetting maintenance: How will your systems be updated in two years?

Ready to build instrumentation software that will ship?

Accendero Software has spent 10 years helping scientific organizations navigate these exact challenges. We've built both web-based platforms that process terabytes and desktop applications that run on ancient laptops. We know when Django scales and when Electron simplifies. Most importantly, we deliver instrumentation software that ships on time and on budget.

Contact us for an instrumentation software architecture consultation.



About Accendero Software

Accendero Software develops custom software solutions that help scientific enterprises accelerate research and enhance discovery. Founded in 2015, we bridge the gap between technical capabilities and scientific needs by combining software engineering expertise with scientific domain knowledge. Our team works directly with clients to build tools that transform complex data into actionable insights, from exploratory research applications to market-ready scientific systems. We focus on delivering solutions that streamline operations while ensuring regulatory compliance. Our US-based team serves clients across the United States, providing high-quality scientific software at competitive rates.

Learn more at <http://www.accendero.com>.