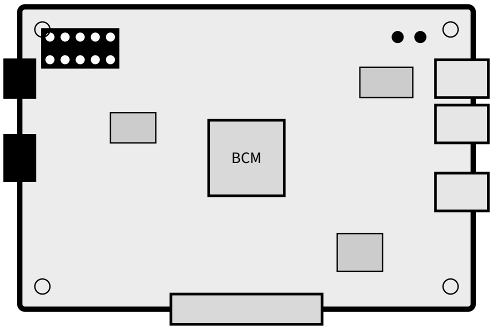
Adam Raźniewski

From Prototype to Product

Building Commercial Devices with Raspberry Pi



Raz Publish

Adam Raźniewski

From Prototype to Product

Building Commercial Devices with Raspberry Pi

Raz Publish

First edition, 2025

ISBN 978-83-977396-0-4

≈

Published by Raz Publish

Preambule

I have spent years building devices based on Raspberry Pi: time attendance machines, camera boards, thermal vision systems, access control devices, even a self-service beer tap (Cybertap).

For a long time, I wished there was a book like this one. A book that didn’t just show the theory, but told the truth about what it really takes to move from an idea to a production-ready product.

I made plenty of mistakes along the way. I chose the wrong boards, I burned Raspberry Pis, I lost months fixing devices. At one point, I was spending eight hours a day repairing customer units because my devices would only last one year — while I was offering a two-year warranty. The biggest lesson? Never rely on SD cards in production.

Despite the failures, I kept going. Over time, I sold more than 5000 of my devices worldwide. Every project taught me something new, and those lessons are what I want to share with you here.

If you want to create devices, put them into the market, and avoid the mistakes I made — this book is for you.

About Author

I’m Adam, and I love to create, program, and build. I’ve always been curious about technology and people, and I enjoy not only designing but also selling what I create.

My first company was born while I was still at university—StudentsCode. Back then, I ran Minecraft servers that surprisingly brought me both a good income and my future wife. Later, I worked at Siemens as a Software Architect, and then at the Texas startup Codenotary, where I grew from Software Developer to Chief Architect over the course of three years.

Eventually, I returned to what had always been close to me: Raspberry Pi. I began producing my own devices, learning the hard way what it means to build for real customers and ship hardware worldwide.

Today, I have designed and delivered thousands of devices based on Raspberry Pi technology, ranging from time attendance systems to access control units, thermal vision cameras, and self-service beer machines. This book collects the lessons I've learned along that journey.

Table of contents

1
Analyzing the problem

Before writing a single line of code or ordering any components, you must crystallize exactly what problem you're solving. This chapter presents a systematic framework for transforming vague business requirements into concrete technical architecture.

Most hardware projects fail not because of technical limitations, but because the problem was never properly defined. I've seen teams spend months building elegant solutions to the wrong problem, only to discover their fundamental assumptions were flawed.

1.1 The Problem Definition Framework

Every successful device project starts with answering one fundamental question: **What do I want to build?**

This seems obvious, but most initial problem statements are too vague for implementation. "I want to build a time attendance system"tells you nothing about authentication methods, data storage, network requirements, or user interface needs.

1.1.1 Step 1: Core Function Definition

Start with a single sentence that captures the essential transformation your device performs:

Example: Time Attendance Registrator

Core Function: Upon user authentication, the device records an attendance event with timestamp and user identity.

Example: Self-Service Beer Machine

Core Function: Authenticate user, calculate beer price, process payment, dispense measured amount of beer, and display product information.

Example: Smart Helmet

Core Function: Stream real-time video and thermal imaging from field operations to command center while maintaining authentication and local storage backup.

1.1.2 Step 2: Core Function Problem Solve Proposal

Once you've defined what your device does, determine how it will accomplish that core function. This step identifies the primary technical challenge and your proposed solution approach.

For each core function, ask: What is the most critical technical problem I need to solve?

Time Attendance Registrator:

- **Primary Challenge:** Reliably identifying users without manual input
- **Solution Proposal:** RFID-based authentication with local database storage
- **Why:** 99% success rate (1% is when employees share the card), low cost, works offline, minimal user training

Self-Service Beer Machine:

- **Primary Challenge:** Accurate volume measurement for variable pricing
- **Solution Proposal:** Flow sensor with real-time price calculation
- **Why:** Enables user-controlled portions while maintaining precise billing

Smart Helmet:

- **Primary Challenge:** Real-time video transmission with network reliability issues
- **Solution Proposal:** Local buffering with automatic upload when connectivity restored
- **Why:** Ensures no data loss during network outages in mission-critical applications

1.1.3 Step 3: Authentication Requirements Analysis

Authentication patterns vary significantly across device types. Identify which pattern fits your project:

Type 1: Local User Authentication (Time Attendance) User authenticates locally for service access, administrator authenticates remotely for reports and management.

Type 2: Cloud-Connected User Authentication (Cybertap) User authenticates locally, but device must communicate with cloud to validate account balance and process transactions.

Type 3: Device-to-Cloud Authentication (Smart Helmet) No user authentication - the helmet itself authenticates to cloud services using unique API key for secure communication.

Time Attendance Registrator - Authentication Options:

Method	Success Rate	Cost	Complexity
RFID Card	99%	Low	Low
QR Code Scan	95%	Low	Medium
Barcode Scan	98%	Low	Medium
Bluetooth Phone ID	85%	Medium	High
Fingerprint	90%	High	High
Face Recognition	85%	High	Very High

Tabela 1.1: Authentication method comparison

Key insight: RFID provides the best balance of reliability, cost, and implementation simplicity. Biometric options sound sophisticated but introduce significant complexity for marginal business value.

1.1.4 Step 4: Interface Requirements

List all required interfaces before considering implementation:

Time Attendance System Interfaces:

- **User Authentication Interface:** RFID reader, status LEDs, audio feedback
- **Administrative Interface:** Web-based user management, report generation

- **Network Interface:** Remote access for data export and system configuration
- **Data Interface:** Local storage with cloud synchronization capability

Self-Service Beer Machine Interfaces:

- **User Interface:** Card reader, balance display, dispense button, volume display
- **Product Interface:** Flow measurement sensor, flow control valve, temperature monitoring
- **Device Interface:** Product identification (each device = one beer type), price per ml configuration
- **Network Interface:** Card balance synchronization, transaction logging, device status reporting

Smart Helmet Interfaces:

- **Video Interface:** 4K camera, thermal imaging sensor, realtime streaming
- **Communication Interface:** Command center authentication, network failover handling
- **Storage Interface:** Local video buffering, automatic upload when connectivity restored
- **Power Interface:** Battery management, charging protocols, power-saving modes

1.2 System Architecture Mapping

Once interfaces are defined, create a system block diagram showing data and control flows.

1.2.1 Time Attendance System Architecture

Required Functions:

- **User identification function:** Read unique identifier from user credential
- **Data processing function:** Match identifier to user database, record timestamp
- **User feedback function:** Indicate successful/failed authentication to user
- **Data storage function:** Store attendance records locally with backup capability
- **Communication function:** Sync data and receive configuration updates
- **Administrative function:** Manage users and generate reports remotely

Critical architectural decisions:

- Local database required for offline operation
- Network interface must handle intermittent connectivity
- Status feedback essential for user confidence
- Administrative interface needed for user management

1.2.2 Self-Service Beer Machine Architecture

Required Functions:

- **Authentication function:** Read and validate user card credentials

- **Balance management function:** Check available balance, deduct costs locally
- **Flow control function:** Start/stop beer dispensing based on user control
- **Measurement function:** Accurately measure dispensed volume for pricing
- **Display function:** Show current balance, volume dispensed, and cost
- **Synchronization function:** Update card balances and transaction logs with central system

Critical architectural decisions:

- Local card balance caching required for offline operation
- Each device represents one beer type (no product selection needed)
- Flow measurement essential for accurate pricing per ml
- Network synchronization needed for card balance updates
- Simplified user interface (no payment terminal required)

1.2.3 Smart Helmet Architecture

Required Functions:

- **Video capture function:** Record high-resolution visible light video
- **Thermal imaging function:** Capture and process thermal camera data

- **Authentication function:** Identify helmet to command center systems
- **Streaming function:** Transmit real-time video feeds with low latency
- **Storage function:** Buffer video locally during network outages
- **Power management function:** Monitor battery, optimize power consumption
- **Communication function:** Handle multiple network types with automatic failover

Critical architectural decisions:

- Video encoding must happen locally to reduce bandwidth requirements
- Local storage essential for network outage scenarios
- Battery monitoring critical for field operation safety
- Dual camera streams require significant processing power
- Multiple network interfaces needed for mission-critical reliability

1.3 Constraint Analysis

Every project has hidden constraints that aren't obvious from the initial requirements. Identify these early:

1.3.1 Physical Constraints

- **Mounting requirements:** Time attendance device must be wall-mountable with standard hardware
- **Chemical resistance:** Beer machine components must resist beer acid and cleaning chemicals
- **Size limitations:** Smart helmet PCB must be compact enough to integrate into helmet design
- **Environmental conditions:** Operating temperature, humidity, vibration resistance
- **Power requirements:** Battery life, charging infrastructure, power consumption limits

1.3.2 Performance Constraints

- **Response time:** Attendance registration under 0.3 seconds (nobody likes to wait), beer dispense start under 2 seconds (somehow people likes beer more that time attendance :))
- **Accuracy requirements:** Payment processing 100% accurate, video quality standards
- **Reliability targets:** 99.9% uptime, mean time between failures
- **Capacity limits:** Concurrent users, data storage requirements, network bandwidth

1.3.3 Business Constraints

- **Cost targets:** Manufacturing cost per unit, development budget limits

- **Time constraints:** Market window, certification timelines
- **Regulatory requirements:** Payment card compliance, safety certifications
- **Support requirements:** Field serviceability, remote diagnostics capability

1.4 Common Analysis Mistakes

Avoid these frequent errors that lead to project failure:

Warning: Feature Creep

Resist the temptation to add "nice to have"features during analysis. Every additional feature exponentially increases complexity, cost, and failure probability. Focus ruthlessly on core requirements.

Other critical mistakes:

- **Underestimating power requirements:** Always measure actual power consumption, don't trust datasheets
- **Ignoring certification requirements:** FCC, CE, safety certifications can add months to timeline
- **Assuming perfect network connectivity:** Design for intermittent and unreliable networks
- **Overlooking field serviceability:** Plan for diagnostics, updates, and repairs from day one

1.5 Moving to Implementation

The next step is translating this analysis into electronic interfaces and component selection. But first, validate your analysis with potential customers.

Show them your interface requirements. Their feedback at this stage costs nothing to incorporate. Changes after you've ordered components cost everything.

Remember: hours spent in problem analysis save weeks in implementation and months in field debugging. The most expensive mistake is building the right solution to the wrong problem.

2
Vision into Electronic Interfaces

Translating functional requirements into specific electronic components is where most hardware projects either succeed or fail spectacularly. This chapter walks through the systematic process of selecting components, using a time attendance system as our example.

The key is methodical evaluation: never choose the first component you find, and never assume compatibility without verification.

2.1 Component Selection Process

Let's build a time attendance system step by step, making real component choices with real tradeoffs.

2.1.1 Step 1: Identify Core Components

From our problem analysis, we need:

- RFID reader for user identification
- Processing unit for database operations
- Network interface for data synchronization
- Local storage (non-volatile)
- User feedback system

2.1.2 Step 2: RFID Reader Selection

Starting with RFID, I search development boards compatible with Raspberry Pi. Two modules dominate the market:

RC522 RFID Module - €3 from Botland (https://botland.store/rfid-modules-tags/6765-rfid-mf-rc522-module-1356mhz-spi-card-and-keychain-5904422335014.html)

PN532 RFID Module - €12-15 from various suppliers

Component comparison:

Specification	**RC522**	**PN532**
Price	€3	€12-15
Library Support (RPi)	Excellent	Limited maintenance
Communication	SPI	I2C/SPI/UART
NFC Support	No	Yes
Range	Standard	Slightly better

Tabela 2.1: RFID module comparison

Critical evaluation questions:

RC522 Analysis:

- **Is this chip widely manufactured?** Yes - RC522 is produced by multiple manufacturers
- **Is it obsolete or current?** The chip is older but still in active production

- **Library support available?** Multiple libraries exist, including my fork for CM4/CM5 kernels (https://github.com/Razikus/razrc522)
- **Cost vs. alternatives?** €3 is competitive for 13.56MHz RFID

PN532 Analysis:

- **Additional capabilities:** NFC support, multiple communication interfaces
- **Library maintenance:** Raspberry Pi libraries are poorly maintained, often outdated
- **Cost impact:** 4-5x more expensive than RC522
- **Production risk:** Limited library support increases development and maintenance burden

Decision: RC522 selected. For time attendance, NFC capability provides no additional value. RC522's lower cost and established library ecosystem outweigh PN532's theoretical advantages.

2.1.3 Step 3: Processing Unit Selection

For Raspberry Pi-based systems, the choice is between standard Pi boards and Compute Modules.

My rule: Never use SD cards in production devices. Period.

This eliminates standard Raspberry Pi boards and directs us to Compute Modules with eMMC storage.

Compute Module 4 - Example: CM4102008

- 2GB RAM, 8GB eMMC, WiFi
- Price: €85 from Farnell
- **Why 2GB RAM?** 1GB is insufficient for modern applications

- **Why 8GB eMMC?** Adequate for OS + application + local database
- **Why not CM5?** Excessive processing power, higher heat generation, unnecessary cost

Critical insight: CM5 lacks hardware H.264 encoding - video applications will consume significant CPU resources for encoding tasks.

2.1.4 Step 4: Carrier Board Decision

Two paths for CM4 integration:

Option 1: Development Carrier Board Waveshare CM4-IO-BASE-C provides cameras, GPIO access, standard interfaces.

Advantages:

- Immediate availability
- No custom PCB design required
- Good for prototyping and small quantities

Disadvantages:

- Requires manual cable assembly
- Not optimized for production assembly
- Higher per-unit cost at scale

Option 2: Custom Production Board Purpose-built carrier with dedicated connectors (example: https://shop.razniewski.eu/p/cm4pb)

Advantages:

- Optimized for production assembly

- TE connectors for RC522, buzzer, indicators
- Lower assembly cost per unit
- Professional appearance

Disadvantages:

- Requires PCB design and tooling investment
- Minimum order quantities
- Longer development timeline

Production volume decision threshold: Above 100 units, custom boards become cost-effective.

2.2 Component Replaceability Strategy

Critical Lesson: Component Obsolescence

We ordered 1000 units after selling our first 100. The Ethernet connector went end-of-life between orders. Fortunately, a replacement existed, but required minor case modifications. Had we used a proprietary micro-USB connector, we would have faced complete redesign.

Replaceability guidelines:

- Choose standard form factors (USB-A, RJ45, standard pin headers)
- Avoid proprietary connectors unless absolutely necessary
- Design footprints that accommodate multiple manufacturer variants
- Document acceptable part substitutions in BOM

2.3 Connector and Cable Strategy

For production devices, cable assemblies determine assembly time and reliability.

Custom Cable Manufacturing: LCSC (https://www.lcsc.com/customcables) provides professional cable assemblies with proper strain relief and connectors.

Example connector choice: TE 3-640621-8 (8-pin connector) - Standard series with multiple variants - Available from multiple suppliers - Established assembly processes

Production assembly goal: Connect RC522 module, connect I/O board, install in case, ship. Minimize field wiring and soldering.

2.4 Camera Integration Considerations

Camera selection depends on processing requirements and driver availability.

USB Cameras with V4L2 Support

- Easiest integration path
- Standard Linux drivers
- No custom kernel modifications required

CSI Cameras

- Higher performance potential
- Requires Raspberry Pi-specific drivers
- CM4 hardware acceleration available

Real Example: Kopin A914 Integration

SmartHelmet project required Kopin A914 microdisplay integration via MIPI-DSI. Custom kernel driver development became necessary. The process was challenging enough with standard Linux development - on Balena platform, documentation was nonexistent, requiring extensive trial-and-error with similar drivers.

Camera selection priority:

1. Standard USB with V4L2 support
2. CSI with established Raspberry Pi drivers
3. Custom hardware only when absolutely necessary

2.5 Microcontroller vs. Raspberry Pi Decision

Not every project requires Raspberry Pi processing power.

Use ESP32 when:

- Simple data collection and transmission
- Low power requirements critical
- Cost optimization paramount
- Real-time response requirements

Use Raspberry Pi when:

- Edge processing required
- Complex user interfaces needed
- Local database operations

- Video/audio processing
- Standard Linux software stack beneficial

Time attendance system evaluation: Requires local database, web interface, and network synchronization. Raspberry Pi appropriate choice.

2.6 Power and Performance Considerations

CM4 vs. CM5 comparison for production:

Specification	**CM4**	**CM5**
Processing Power	Adequate for most apps	Excessive for many cases
Heat Generation	Manageable	Requires active cooling
Hardware H.264 Encoding	Yes	No (CPU-based)
Power Consumption	Lower	Higher
Cost	Lower	Higher

Tabela 2.2: CM4 vs CM5 production comparison

Recommendation: CM4 remains optimal for most production applications unless specific CM5 features are required.

2.7 Validation and Testing Strategy

Before committing to large quantities:

1. **Prototype with development boards:** Verify functionality with Waveshare or similar carrier boards

2. **Test component availability:** Order small quantities from multiple suppliers
3. **Validate software libraries:** Ensure drivers work with your kernel version
4. **Thermal testing:** Verify thermal performance in your enclosure
5. **Production assembly test:** Build 5-10 units with production processes

2.8 Documentation for Production

Create these documents before scaling:

- **Bill of Materials (BOM):** Include acceptable substitutions
- **Assembly instructions:** Step-by-step with photos
- **Test procedures:** Functional verification steps
- **Connector pinouts:** Cable assembly documentation
- **Software configuration:** Deployment and setup procedures

2.9 Moving Forward

With components selected and interfaces defined, the next step is building the "spider prototype- a functional system using development boards and jumper wires to validate the complete system integration before committing to custom PCB design.

Remember: the goal is not perfect component selection, but systematic evaluation that minimizes risk while maintaining cost targets. Every component choice creates dependencies - cho-ose components that provide multiple supply sources and es-tablished ecosystem support.

3
Build a "spider"prototype

You've analyzed the problem and selected components. Now comes the critical validation phase: proving your concept actually works before committing to PCB design and production tooling.

This is the transition from TRL 3 (experimental proof of concept) to TRL 4 (technology validated in lab). The "spider"prototype gets its name from the web of jumper wires connecting components - it looks chaotic but serves a crucial purpose.

3.1 The Technology Readiness Level Context

Understanding where you are in the development process prevents premature optimization:

TRL 1-2: Basic principles and concept formulation (completed in analysis phase) **TRL 3:** Experimental proof of concept (what we're building now) **TRL 4:** Technology validated in lab environment (our goal for this chapter) **TRL 5-6:** Validation in relevant/operational environment (future chapters) **TRL 7-9:** System demonstration and proven operation (production phase)

3.2 Spider Prototype Objectives

The spider prototype must validate every critical function:
For our time attendance system:

- RFID reader successfully captures UID
- CM4 maintains stable internet connection
- System executes REST API calls reliably
- Buzzer provides audio feedback
- GPIO controls function correctly
- Data persistence works (file or database)
- Web interface displays collected data

3.3 Minimal Software Requirements

Write the simplest possible code that exercises every function:

Basic Validation Script

```python
import time
import requests
from razrc522.rfid import RFID
from razrc522.easyrfid import EasyRFID, EasyRFIDUIDMode, EasyRFIDAuth
reader = RFID(antenna_gain=7, logger=None)
easyRFID = EasyRFID(reader, mode=EasyRFIDUIDMode.HEX)
def test_network():
    try:
        response = requests.get('https://httpbin.org/get', timeout=5)
        return response.status_code == 200
    except:
        return False
def read_rfid_card():
    try:
        convertedUID, rawUID = easyRFID.wait_and_select()
        block = 0
        # Standard MIFARE 1k key
        authorized = easyRFID.authorize(EasyRFIDAuth.AuthB,
                                                               [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF],
                                                               rawUID, block)
        if authorized:
            readed = easyRFID.read_block(block)
            ....
...
```

3.4 Critical Validation Questions

Run your spider prototype and systematically evaluate:

3.4.1 Basic Functionality

- **Does it work at all?** Can you complete the full cycle without errors?
- **How cumbersome is interaction?** Does RFID reading require multiple attempts?
- **Response time:** How fast does the system respond to card presentation?
- **Reliability:** Test the same operation 50 times - how many succeed?

3.4.2 Environmental Factors

- **Heat generation:** Does the CM4 require cooling under normal operation?
- **Power consumption:** Measure actual current draw vs. specifications
- **Voltage compatibility:** Verify 3.3V GPIO levels work with your peripherals
- **EMI sensitivity:** Do nearby phones or WiFi networks cause interference?

3.4.3 Persistence Testing

- **24-hour test:** Does the system work reliably after running overnight?

- **1-hour stress test:** Continuous operation with frequent card reads
- **Network interruption:** How does the system behave when WiFi disconnects?
- **Power cycle recovery:** Does it restart cleanly after power loss?

Critical Voltage Warning

Raspberry Pi GPIO operates at 3.3V, not 5V. Connecting 5V signals to GPIO pins will damage your CM4. Always verify peripheral voltage requirements and use level shifters when necessary.

3.5 Spider Prototype Assembly

Option 1: Individual Components

Required components for time attendance validation:

- CM4 on development carrier board (Waveshare CM4-IO-BASE-C or similar)
- RC522 RFID module
- Buzzer (3.3V compatible)
- LED for status indication
- Breadboard and jumper wires
- RFID cards/tags for testing

Connection diagram:

```
RC522 -> CM4 GPIO
SDA   -> GPIO 8  (SPI CE0)
SCK   -> GPIO 11 (SPI CLK)
MOSI  -> GPIO 10 (SPI MOSI)
MISO  -> GPIO 9  (SPI MISO)
RST   -> GPIO 25
3.3V  -> 3.3V
GND   -> GND

Buzzer -> GPIO 18
LED    -> GPIO 24 (with current limiting resistor)
```

Option 2: Complete RFID Solution

For faster prototyping, use the complete CM4 RFID board (https://shop.razniewski.eu/p/cm4pb) which includes CM4, RFID reader, and all necessary interfaces pre-integrated. This eliminates wiring complexity and reduces validation time to software testing only.

3.6 Iteration and Refinement

Spider prototypes exist to fail safely. Expect multiple iterations:

Common first iteration failures:

- RFID reading inconsistent (adjust antenna positioning)
- Network timeouts (implement retry logic)
- GPIO conflicts (verify pin assignments)
- Power supply inadequate (check current requirements)

Performance optimization targets:

- RFID read success rate > 95%

- Response time < 2 seconds
- Network operation timeout < 5 seconds
- System uptime > 24 hours without intervention

Material interference testing: Test RFID scanning through different materials, particularly those you plan to use for the final enclosure:

- Plastic enclosure materials (ABS, polycarbonate)
- Metal proximity effects (mounting screws, brackets)
- Glass or acrylic cover materials
- Varying material thicknesses

3.7 Component Validation Results

After spider prototype testing, you'll know definitively:

Keep these components if:

- Functionality works reliably in spider configuration
- Performance meets your requirements
- Heat generation is manageable
- Cost targets are achievable

Reconsider components if:

- Reliability issues persist after optimization attempts
- Performance significantly below expectations
- Integration complexity exceeds development budget
- Alternative components offer substantial advantages

3.8 Decision Point: PCB or Software First

With validated spider prototype, choose your next development path:

PCB Design First:

- Hardware functionality proven stable
- Mechanical constraints well understood
- Production timeline pressure
- External PCB design resources available

Software Development First:

- Complex software requirements
- User interface needs refinement
- Integration with external systems required
- Software team available in-house

My typical approach: Software development while PCB design proceeds in parallel. Software can be developed and tested on spider prototype while PCB layout is completed by external partners.

3.9 External PCB Design Considerations

Most hardware entrepreneurs benefit from partnering with PCB design specialists rather than learning complex layout tools.

First PCB reality check: Your first PCB will not be ready for final enclosure integration. Accept this from the start.

At this point, consider your enclosure strategy by reviewing the Appendix resources:

- **Kradex:** Standard enclosures for cost-effective solutions
- **Cubic Inch:** Custom 3D printed enclosures for unique requirements

Think about placement and constraints, but don't optimize yet:

- How will components be positioned relative to each other?
- What connectors need external access?
- Where will status indicators be visible?
- How will the RFID antenna be oriented?

Mechanical constraints will change as you iterate. The PCB will evolve significantly between first prototype and production version.

Prepare for external PCB design:

- Detailed pinout documentation from spider prototype
- Basic mechanical requirements (board size, connector positions)
- Performance requirements (signal integrity, EMI considerations)
- Manufacturing constraints (assembly capabilities, test points)

Iteration expectation: Plan for 2 PCB iterations minimum. The first validates electrical functionality. The second optimizes for mechanical integration and manufacturing.

3.10 Moving to Implementation

Your spider prototype has validated the core technology. You now have:

- Proven component functionality
- Working software foundation
- Performance baselines
- Integration challenges identified
- Development path clarity

The next phase involves either developing production software on your proven hardware platform or translating the spider prototype into a proper PCB design that maintains the validated functionality while enabling efficient manufacturing.

Remember: the spider prototype's job is to fail fast and fail cheap. Every failure discovered here saves weeks of debugging production hardware later.

4 Write a software

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

5 Key checklist for production code

WRITING CODE FOR PRODUCTION devices is fundamentally different from hobbyist projects or internal tools. When your code ships to customers who expect commercial-grade reliability, every design decision becomes critical. A single oversight can result in thousands of field service calls or, worse, a completely failed product launch.

After shipping over 5,000 devices and dealing with the consequences of poor production code decisions, I've learned these lessons the hard way. This chapter distills the most critical considerations for production-ready code.

5.1 Remote Update Capability

Always implement remote update functionality from day one.

You will need to update your devices. Not might need—will need. Whether it's fixing a critical bug, adding a customer-requested feature, or simply changing the font size in the UI, the ability to update deployed devices remotely is non-negotiable.

Real Crisis: Time Calculation Bug

I discovered a critical bug in my time attendance recorder that incorrectly calculated overtime hours in specific edge cases—when employees clocked in exactly at midnight during daylight saving time transitions. The bug affected payroll calculations for dozens of companies. Without remote update capability, I would have had to physically visit hundreds of deployed devices or face potential lawsuits and complete reputation destruction. The remote update saved my business.

Minor Issues Add Up

Even seemingly trivial updates matter. A client once threatened to cancel a 500-unit order because they couldn't read the status display—the font was too small for their environment. Without remote update capability, I would have faced the choice between losing the client or manually visiting 50+ already-deployed devices to flash new firmware.

Implementation considerations:

- Use cryptographically signed updates to prevent tampering (e.g., using GPG signatures, x.509 certificates. Could be even git pull based)
- Implement rollback mechanisms for failed updates (e.g., using versioned firmware, A/B OTA updates)
- Design your update system before writing application code

- Test the update process extensively—it's harder to fix than the original code
- **Consider Balena Cloud:** Provides enterprise-grade remote update capability out of the box

5.2 SSH Access Strategy

Plan your support access carefully, considering both security and customer constraints.

SSH access for support is essential, but many corporate environments block SSH traffic or restrict open ports. You need a support strategy that works within your customers' security policies.

Options to consider:

1. **Custom SSH port:** Use non-standard ports (e.g., 2222, 8022) to avoid basic port blocking
2. **Customer-triggered access:** Implement a mechanism where customers can temporarily enable SSH when support is needed
3. **Reverse SSH tunnels:** Device initiates connection to your support server
4. **VPN solutions:** Tools like Balena or custom VPN implementations
5. **Balena Cloud SSH:** Provides secure SSH access through their cloud infrastructure, bypassing firewall restrictions

Real-World Constraint

Enterprise customers often have strict firewall policies. Plan your support access strategy during the design phase, not when you're desperately trying to troubleshoot a failing device in production.

5.3 Operating System Selection

Choose your OS foundation carefully—this decision affects everything else.

For Raspberry Pi devices, you have several paths:

5.3.1 Custom Raspbian Build

Build your own minimal OS using pi-gen: `https://github.com/RPi-Distro/pi-gen`

Advantages:

- Complete control over included packages
- Smaller attack surface
- Faster boot times
- Reproducible builds

Disadvantages:

- Significant initial setup time
- You're responsible for security updates
- Limited ecosystem support

5.3.2 Balena OS Alternative

Balena provides a complete IoT device management platform with both paid and open-source options.

Balena Cloud (Paid):

- Out-of-the-box device management
- Automatic updates and monitoring
- Professional support
- Docker-based application deployment
- **Built-in remote update capability:** Push updates to devices worldwide with a single command

OpenBalena (Open Source):

- Self-hosted device management
- No ongoing costs
- Full control over infrastructure

Balena Gotchas

Certification Issues: Balena certificates can expire, potentially breaking device connectivity.
Always-On VPN: Even when disabled, the VPN infrastructure remains present, which some security-conscious customers reject.
Driver Limitations: Custom hardware drivers can be challenging to integrate into Balena's Docker ecosystem.
Docker Overhead: Additional resource usage may impact performance on resource-constrained devices.

Despite these limitations, Balena works exceptionally well for most use cases and can significantly reduce development time.

5.4 Multiple Connectivity Options

Always provide at least two ways to connect to your device.

Network connectivity is the most common point of failure in deployed devices. Customers will find themselves in situations where their primary connection method fails.

Proven combinations:

- **Ethernet + WiFi:** Most reliable for stationary installations
- **WiFi + Bluetooth:** Useful for mobile or temporary deployments
- **Ethernet + Bluetooth:** Good for environments with WiFi restrictions

In my time attendance device "Skryba,"I implemented WiFi + Bluetooth networking. When customers had WiFi connectivity issues, they could still configure and operate the device via Bluetooth using a mobile app.

5.5 Zero-Configuration User Experience

Customers are not developers—design for plug-and-play operation.

Your device should work immediately upon power-up, requiring minimal configuration. Implement zero-configuration networking protocols and automatic device discovery.

Key principles:

- Default to automatic IP configuration (DHCP)
- Broadcast device presence (mDNS/Bonjour)

- Provide web-based configuration interface
- Use QR codes for easy mobile device pairing
- Implement status LEDs for non-technical troubleshooting

5.6 Power Supply Considerations

Assume inconsistent power quality and implement atomic operations.

Customer power supplies are often unreliable. I once had a client whose device constantly shut down unexpectedly due to insufficient power supply capacity. Your code must handle sudden power loss gracefully.

Critical practices:

- Use atomic file operations for critical data
- Implement transaction logs for multi-step operations
- Design state machines that can recover from any interruption
- Consider using a supercapacitor for graceful shutdown detection

5.7 Hardware Boundary Enforcement

Explicitly limit supported hardware configurations.

If you support HDMI output, specify exactly which resolutions you support. In my experience, stating ï920×1080 onlyp̈revents countless support issues with customers using unusual display configurations.

Common boundaries to enforce:

- Display resolutions and refresh rates
- Network interface speeds
- Storage device types and sizes
- USB device classes
- Camera resolutions and frame rates

5.8 Automatic Recovery Mechanisms

Implement comprehensive auto-recovery for all failure modes.

Customers expect devices to handle problems automatically without manual intervention.

Recovery strategies:

- **Application crash:** Automatic restart with exponential backoff
- **Network disconnection:** Automatic reconnection with fallback methods
- **Memory exhaustion:** Automatic reboot when available memory drops below threshold
- **Disk full:** Automatic log rotation and cleanup of temporary files

Watchdog Implementation

```bash
#!/bin/bash
# Simple application watchdog
while true; do
    if ! pgrep -f "my_application" > /dev/null; then
        echo "Application not running, restarting..."
        /opt/myapp/start_application.sh
        sleep 30
    fi
    sleep 10
done
```

5.9 Credential Security

Never hardcode credentials that could compromise the entire application.

This includes API keys, database passwords, encryption keys, and any authentication tokens. Hardcoded credentials create a single point of failure that affects all deployed devices.

Secure alternatives:

- Device-specific certificates generated during manufacturing
- Runtime credential provisioning through secure channels
- Hardware security modules (HSM) for key storage
- Encrypted credential storage with device-unique keys

5.10 Local Security

Secure all endpoints, even on local networks.

Never expose password-unprotected endpoints, even on supposedly "trusted"local networks. Local networks are often less secure than public internet connections.

Minimum security measures:

- Authentication required for all administrative functions
- HTTPS for all web interfaces, even local ones
- Input validation on all API endpoints
- Rate limiting to prevent brute-force attacks

5.11 The Embedded Golden Rule

Simpler is always better in embedded systems.

Every additional feature, dependency, or complexity point increases the probability of failure. When in doubt, choose the simpler solution.

Complexity indicators to avoid:

- Multiple programming languages in one project
- Complex dependency chains
- Custom protocols when standard ones exist
- "Smart"features that aren't explicitly requested
- Premature optimization

This principle has saved me more time and customer relationships than any other rule. When facing a choice between an elegant complex solution and a simple working solution, always choose simple.

5.12 Two paths

Always have at least 2 paths - development and production.
Push into development first.

5.13 Pre-Deployment Checklist

Before shipping any code update, verify on development path:

1. Remote update mechanism tested and working
2. Support access method confirmed with customer IT
3. All hardcoded values removed or configurable
4. Auto-recovery mechanisms tested with actual failures
5. Security endpoints password-protected
6. Zero-configuration setup tested by non-technical users (ask your relative)
7. Power interruption recovery tested
8. Multiple connectivity methods verified

Following these principles won't eliminate all production issues, but it will dramatically reduce the frequency and severity of field problems. More importantly, it will give you the tools to fix issues remotely when they do occur.

Remember: production code isn't just about making features work—it's about making them work reliably in environments you can't control, for users who aren't experts, with hardware that will inevitably fail in unexpected ways.

And believe me. It's a mess.

6
Key checklist for production hardware

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

7
From spider into first 5 PCB

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

8
Test the process

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

9
Certification

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1]Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

10

First 100 boards

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

11 Document process

LOREM IPSUM DOLOR SIT AMET, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesentimperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet antelobortis[1] sollicitudin. Praesent blandit blandit mauris. Praesent lectustellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia loremsit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

[1] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

12 Repeat

Scaling from your first successful product to a sustainable business requires systematic repetition of proven processes. After shipping your first 100 boards, the temptation is to celebrate and move on to the next idea. Resist this urge.

The real profit comes from iteration and refinement of existing products. Each production run teaches you something new about manufacturing, quality control, and customer needs.

12.1 The Iteration Mindset

Every production run is an experiment. Document what works and what doesn't. Your goal is to reduce the time between identifying a market need and delivering a solution.

> *"The first version of your product will be wrong. The second version will be less wrong. By the tenth version, you might actually have something customers want to pay for."* — Manufacturing wisdom

12.2 Version Control for Hardware

Unlike software, hardware versions are expensive. Each PCB revision costs money and time. Establish clear criteria for when to rev your board:

- Critical bugs that affect more than 5% of units
- Component obsolescence forcing a redesign
- Cost reduction opportunities greater than 15%
- Customer-requested features with proven demand

12.3 Production Checklist

Before each production run, verify these items:

1. **Component availability** — Check lead times for all parts
2. **Assembly house capacity** — Confirm production slots
3. **Test procedures** — Update based on field failures
4. **Packaging** — Ensure adequate protection for shipping
5. **Documentation** — Update assembly drawings and BOMs

12.4 Common Pitfalls

12.4.1 The Feature Creep Trap

Customers will request endless features. Not all feedback is worth implementing. Ask yourself:

- Does this solve a real problem for multiple customers?

- Can we implement it without major redesign?

- Will this feature increase or decrease reliability?

12.4.2 Cost Optimization Timing

Don't optimize costs too early. Wait until you have:

- Stable hardware design (no major revisions for 6 months)

- Predictable demand (at least 3 consecutive successful production runs)

- Clear understanding of failure modes

12.5 Code Quality in Production

Your software must be bulletproof. Here's a minimal checklist for production code:

Production Validation Script

```bash
#!/bin/bash
# Pre-deployment checklist
echo "Running production code validation..."

# Check for debug statements
if grep -r "console.log|print|debug" src/; then
    echo "ERROR: Debug statements found!"
    exit 1
fi

# Verify error handling
if ! grep -r "try|catch|except" src/; then
    echo "WARNING: No error handling found"
fi

# Check for hardcoded values
if grep -r "192.168|localhost|127.0.0.1" src/; then
    echo "ERROR: Hardcoded network values found!"
    exit 1
fi

echo "Code validation passed"
```

12.6 Remote Updates

Caution: Remote Updates

Remote update capability is essential but dangerous. Always implement:

- Rollback mechanism for failed updates
- Cryptographic signature verification
- Staged rollout (update 10% of devices first)
- Manual recovery method (physical button or jumper)

A failed update that bricks devices in the field will destroy your reputation overnight.

12.7 Scaling Challenges

12.7.1 Supply Chain Management

As you scale beyond 1000 units per year, component sourcing becomes critical:

- Establish relationships with multiple suppliers
- Monitor component lifecycle status monthly
- Maintain 3-6 months of critical component inventory
- Design alternative footprints for key components

12.7.2 Quality Assurance

Your test coverage must scale with production volume:

Production Volume	**Test Coverage**	**Sample Size**
1-10 units	100% functional test	All units
11-100 units	Functional + burn-in	All units
100+ units	Statistical sampling	10% + outliers

Tabela 12.1: Recommended testing strategy by volume

12.8 When to Stop

Know when to discontinue a product. Clear exit criteria prevent endless resource drain:

- Declining sales for 3 consecutive quarters
- Component obsolescence requiring major redesign
- Competitor products offering 50% better price/performance
- Support costs exceeding gross profit margin

The hardest decision in hardware is knowing when to stop improving and ship, and knowing when to stop shipping and move on.

12.9 Building Your Next Product

Use lessons from your first product to accelerate the second:

1. Reuse proven hardware blocks (power supplies, communication interfaces)

2. Maintain design consistency (connector types, mounting holes)
3. Leverage existing supplier relationships
4. Apply learned failure modes to new designs

The second product should take 60% of the time the first one did. The third should take 40%. If it's taking longer, you're not learning from your mistakes.

12.10 The Long Game

Building a sustainable hardware business requires patience and persistence. Your goal is not just to ship one successful product, but to build repeatable processes that can generate multiple successful products over time.

Document everything. Every mistake, every successful solution, every supplier relationship. This knowledge becomes your competitive advantage.

Success in hardware comes not from building one perfect product, but from building an imperfect product, learning from its flaws, and systematically improving with each iteration.

The companies that win are those that can iterate fastest while maintaining quality. Speed matters, but consistency matters more.

Appendix: Essential Resources

This appendix contains resources from my personal experience building and shipping thousands of hardware devices. These are my actual partners and suppliers—companies I've worked with directly and can recommend based on real production scenarios, not theoretical evaluations.

Every entry in this list has been tested through multiple projects and production runs. These relationships were built over years of trial and error, successful deliveries, and occasional failures that taught valuable lessons.

12.11 Component Sourcing

Digi-Key (https://www.digikey.com) Industry standard for electronic components. Excellent search functionality, reliable stock information, and fast shipping. Use for prototyping and small production runs.

Mouser Electronics (https://www.mouser.com) Similar to Digi-Key with slightly different inventory. Good for cross-referencing part availability and pricing.

TME (https://www.tme.eu) European distributor with competitive pricing and good local support in Poland. Useful for cost optimization in European markets.

Botland (https://botland.store) Polish distributor specializing in development boards and maker-friendly components. Good for Raspberry Pi accessories and sensors.

Waveshare (https://www.waveshare.com) Direct manufacturer of Raspberry Pi accessories and industrial modules. Quality varies, but pricing is competitive for development boards and displays.

12.12 Manufacturing Partners

12.12.1 PCB Manufacturing

JLCPCB (https://jlcpcb.com) Exceptional value for PCB manufacturing and assembly. Their SMT assembly service can populate basic components for prototypes. Critical for rapid iteration during development.

LCSC Custom Cables (https://www.lcsc.com/customcables) Outstanding custom cable manufacturing. Clean documentation, reasonable pricing, and reliable quality. Essential for professional-looking prototypes and production units.

12.12.2 Mechanical Manufacturing

Fullbax Formy (https://fullbax-formy.pl/en/) Premier injection molding partner with established China manufacturing relationships. Use for production volumes above 1000 units where custom enclosures are required.

Cubic Inch (https://cubicinch.pl/en/home/) Leading MJF (Multi Jet Fusion) 3D printing service. Superior surface finish and mechanical properties compared to FDM printing. Ideal for functional prototypes and low-volume production parts.

Kradex (https://www.kradex.com.pl/?lang=en) Extensive catalog of ready-made enclosures. Avoid custom tooling costs by designing your PCB to fit standard enclosures. Significant cost savings for smaller production runs.

12.13 Raspberry Pi Compute Module Solutions

CM4/CM5 Production Base Board (https://shop.razniewski.eu/p/cm4pb) Custom base board designed for production use with integrated RFID, RTC, and standard interfaces. Eliminates need for custom carrier board development in many applications.

CM4/CM5 Programmer (https://shop.razniewski.eu/p/cm4prog) Essential tool for Compute Module development. Enables reliable flashing and recovery of CM4/CM5 modules during development and production.

12.14 Cloud Infrastructure

Hetzner (https://www.hetzner.com) European cloud provider with excellent price-to-performance ratio. Particularly strong for applications requiring GDPR compliance and European data residency.

Balena Cloud (https://www.balena.io/cloud) Specialized IoT device management platform. Provides over-the-air updates, remote access, and fleet management capabilities. Consider for projects requiring remote device management.

12.15 Software Development

Pi-Gen (https://github.com/RPi-Distro/pi-gen) Official Raspberry Pi OS build system. Essential for creating custom, reproducible OS images for production devices. Steep learning curve but provides complete control over the software stack.

12.16 Evaluation Criteria

When selecting suppliers and partners, prioritize these factors:

1. **Reliability:** Consistent delivery times and quality standards
2. **Communication:** Responsive technical support and clear documentation
3. **Scalability:** Ability to handle growth from prototype to production volumes
4. **Geographic proximity:** Shorter supply chains reduce risk and shipping costs
5. **Technical capabilities:** Understanding of your specific requirements and constraints

12.17 Cost Optimization Strategy

Development Phase: Use premium suppliers (Digi-Key, Mouser) for fast iteration and reliable components.

Pre-production: Evaluate cost alternatives (TME, direct manufacturers) while maintaining quality standards.

Production: Establish relationships with multiple suppliers to ensure continuity and competitive pricing.

12.18 Warning: Supplier Risks

Critical Supplier Considerations

Single-source dependencies: Never rely on a single supplier for critical components, especially for production volumes.
Quality variability: Lower-cost suppliers may have inconsistent quality. Always order samples before committing to large quantities.
Lead time volatility: Global supply chain disruptions can extend lead times from weeks to months. Plan accordingly.
Minimum order quantities: Production suppliers often require larger minimum orders than development-phase suppliers.

Remember: These resources represent years of trial and error. Each has been validated in real production scenarios, but your specific requirements may lead to different optimal choices. Use this as a starting point, not a definitive solution.

Learn to build production-ready devices with Raspberry Pi Compute Module

This practical guide takes you from prototype to market-ready product. Based on real experience from building and selling over 5,000 devices worldwide.

What you'll learn:

- Convert your vision into electronic interfaces
- Choose the right peripherals and avoid costly mistakes
- Build reliable prototypes and production PCBs
- Write production-ready software with remote updates
- Navigate certification processes
- Scale from prototype to first 100 devices

Perfect for: Engineers, makers, and entrepreneurs who want to turn their Raspberry Pi projects into commercial products.

Raz Publish

Adam Raźniewski