Strengthen your Research by Telling Its Software Story

4 - Template Walkthrough

EVERSE Network Training, Michael Sparks (Nov 2025)

Agenda

First Part

- 1 Introduction and context
- 2 What are Research Software Stories?
- 3 Why write them
- ***4 How we write them - Template walkthrough*** **<– We are here**

Second Part

- Guided Writing Session
- Discussion, Q&A
- Support Materials https://tinyurl.com/EVERSE202511-RSS

How we write them - Template Walkthrough

- Our template provides a scaffold - a way to externalise context you already know but may rarely write down.

Now that we've talked about what Research Software Stories are and why writing one is worthwhile, I want to turn to something very practical: how to actually write the story itself. And the good news is that the structure already exists. The RSQKit template gives you a clear, predictable set of headings that guide you through the process step by step. They also signify to readers a common language of where to look for specific details.

What We're Doing in This Walkthrough

- Move through the template *in order*
- Explain *what each section is for*
- Explain *why it matters to research*
- Show *what kinds of answers belong there*

What I'd like to do now is walk through that template with you, section by section, and explain not just what goes where, but why each part matters and how it helps your research.

First Impression: "This Looks Like a Lot"

But each section is small. And each exists because *research benefits* from that piece of clarity.

- The problem
- The people
- The technology
- The practices
- The tools
- FAIR & openness
- Documentation
- Sustainability

You may look at the template for the first time and think, "This is quite a lot." But once you understand the purpose behind each section, it becomes far easier. The aim is not to produce a perfect document on the first try; it is to give you a scaffold

so that you can externalise knowledge that otherwise stays buried in your head.

The Template as a Thinking Tool

You don't write a polished document. You let the headings ask you the right questions.

The template doesn't demand perfection — it simply prompts the right questions.

Let's begin at the top.

Section 1: The Problem

What scientific, analytical, or operational problem does the software solve?

The Problem

The first section, "The Problem", looks deceptively simple. It asks you to describe the problem your software is trying to solve, ideally with a short one-line subtitle summarising it. But this is more meaningful than it might appear. In research environments, people often dive straight into explaining code or functionality without ever stating clearly what the software is for. This section forces you to step back and articulate the scientific, analytical or operational problem that motivated the software in the first place.

What To Write Here

The bottleneck, data challenge, manual workflow, reproducibility issue, or fragile process that triggered the need for software.

The wording here should be straightforward. You're describing the challenge your community faces - the bottleneck, the inefficiency, the barrier to reproducibility, the data handling issue, the workflow pain point. Every piece of research software exists because something was too manual, too repetitive, too fragile or too complex to manage without computational help. This is your chance to explain that in plain language.

Why "The Problem" Matters

- Sets scope
- Defines motivation
- Prevents misaligned expectations
- Anchors every other section

The reason this matters is that the problem defines the project. It determines the scope. It sets expectations. Without a clear statement of the problem, it becomes very difficult for collaborators or contributors to understand what the software is not trying to solve. And that is just as important. A Research Software Story begins here because every other decision flows from this point.

Section 2: User Community

Who builds the software and who uses it?

User Community

The next section focuses on the community - the people who build the software and the people who use it. Again, the template asks for a brief subtitle to orient the reader, and then expands into details about development and users.

What This Section Reveals

- How development is organised
- Who relies on the software
- Range of skills and backgrounds
- Where the project sits socially in the research ecosystem

This is where you explain how your project is organised. Is it a small team working on a narrow problem? Is it a large collaborative codebase maintained by several groups across institutions? Is the software developed by one researcher in their spare time? All of these are valid models, and writing them down helps others understand why the project works the way it does.

You also describe the user community. Who depends on this software? How experienced are they? Are they domain experts with minimal programming experience, or computational scientists comfortable reading code? Are they students, postdocs, or senior researchers? Knowing the audience helps readers interpret the rest of the story correctly. A project used by first-year PhD students requires different onboarding expectations than one used by instrumentation specialists on a large experiment.

Why This Section Matters

It shows:

- Resource realities
- Risks or over-reliance
- Expected expertise
- How people interact with the code

This section also supports self-reflection. Many research groups don't think of themselves as "communities", but they are. Writing this out can reveal whether the project is effectively resourced, whether the user base is broader than expected, or whether development relies too heavily on one person. It can show gaps or risks that aren't visible when you're focused solely on the code.

Section 3: Technical Aspects

Describe what the software *is*, technically.

Technical Aspects

After establishing the human context, the story turns to the technical context about what the software actually is.

What Goes Here

- Software tier (analysis, prototype, infrastructure…)
- Languages
- Codebase size, age, complexity
- Deployment environment

This section asks you to describe the type of software you're dealing with - whether it's analysis code, prototype tools, research infrastructure, or something more service-oriented. It then leads you into a discussion about languages, complexity, history, deployment requirements and other technical aspects.

Why This Helps Readers

It allows researchers, RSEs, and funders to understand:

- Maturity
- Fragility
- Scale
- Effort required

The aim here is not to overwhelm the reader with detail, nor is it to list every dependency. Instead, it is to give a grounded picture of the environment your software lives in. For example, if your software is a high-performance analysis tool that requires specific cluster configurations or GPU access, that matters. If it is a small library that runs on any laptop, that matters too. If the codebase is twenty thousand lines long and has been in development for six years, that conveys something different from a brand-new script written for a single paper.

This section also helps collaborators make informed decisions. An RSE reading this will understand immediately whether they're dealing with a fragile prototype that needs restructuring, or a mature system that requires careful handling. A researcher reading this will understand whether the project is scaling towards service status, or whether it remains experimental. And a funder reading this will understand the level of investment needed to sustain it.

Section 3a: Libraries & Systems

List the frameworks, libraries, domain tools and formats the software relies on.

The related section on "Libraries and Systems" is simply an extension of this. It asks

you to name the ecosystem your software relies on - not every detail, but the major frameworks, data formats, scientific libraries or domain-specific systems. This is especially helpful for people outside your field who may have heard the names but not know how they fit into your work.

Section 4: Software Quality Practices

How the software is developed - not the code, but the *culture* around the code.

Software Quality Practices - this is how the software is developed - not the code, but the *culture* around the code.

What To Include

- Version control
- Licensing
- Contribution processes
- Testing
- CI
- Release workflow

The next section invites you to describe the software practices used in development. This includes version control, licensing, contribution processes, testing, continuous integration, code review, release strategies and other practices. Many research groups underestimate how much they already do. Even if development feels informal, chances are that some practices exist. Writing them down helps make them visible.

Why Practices Matter

Practices show:

- Reliability
- Sustainability
- Trustworthiness
- Expectations for contributors

This section is important because it captures how the software is developed, not just what it does. These practices shape reliability, sustainability and trust. They also tell collaborators what to expect when they contribute. If you have a formal review system, newcomers will understand that changes are checked before merging. If you don't, that's also important to state, because it may affect how others approach work.

This section also highlights the benefits the developers themselves have gained from using good practices. Perhaps using version control made it easier to collaborate with students. Perhaps a simple testing workflow prevented regressions during paper submission deadlines. Whatever the case, this section situates the project in a broader culture of research software quality.

Section 5: Developer Community

This section is not about "who uses it" - it's about the developers.

What Goes Here

- How new developers join
- How new users learn
- What typical user journeys look like (user stories)

The template includes a second "Community" heading focusing not just on the developers and users as before, but on adoption and onboarding. This is the space where you describe how new developers join, how new users learn the software, and what the key user stories look like.

Why This Section Is Valuable

It tells newcomers:

- What to expect
- How to start

- Where guidance exists
- How mature or accessible the project is

This is one of the most valuable sections in the entire template. Every project has repeat patterns - typical user types who want to achieve certain things. Perhaps you have students who want to run a standard workflow. Perhaps you have senior researchers who want to adjust parameters. Perhaps you have external collaborators who want to use a subset of features. Describing these user stories helps make the purpose of the software even clearer.

In this section, you also describe your onboarding process. Do new users start with example notebooks? Do new developers pair-program with you? Does the project rely on ad-hoc mentoring? Knowing how people begin tells a newcomer what to expect and gives a funder insight into project maturity.

Section 6: Tools

The quality-enabling tools that support development and maintenance.

Tools

This section asks you to describe the tools that support software quality.

What To Describe

- Linters and formatters
- CI systems
- Profilers
- Documentation tooling
- Containerisation
- Workflow orchestration

This includes things like: linters, formatters, profiling tools, documentation generators, workflow engines, CI systems, container builders, or anything else that helps maintain or assess the software.

Why Tools Matter

Tools show the project is actively maintained and designed for reliability.

What matters here is not to list everything, but to explain how these tools help your project. If your project uses pre-commit hooks to prevent accidental errors, say so. If you rely on containers to avoid environment drift, explain why. If your CI system runs regression tests, mention it. This section shows that the project is being actively stewarded rather than left to drift.

Link to RSQKit

Many tools can also map directly to RSQKit tasks. This helps identify which pathways the project is already aligned with.

This can also links directly to RSQKit, because many of these tools correspond to tasks described in RSQKit's quality pathways. A reader who understands which tools you use will immediately see which practices are in place and which might need further support.

Section 7: FAIR & Open

How the project aligns with FAIR software principles and open development practice.

This section brings together FAIR principles and open-source practice. FAIR is increasingly central to research software, but you do not need to tick every box. The story simply asks you to explain how far along you are.

FAIR: What To Address

- Public repository
- Licence
- Citation metadata
- Release structure
- Registry/catalogue presence

- Issue tracking
- Code review openness
- Contribution visibility
- Transparency of decisions

Do you have a public repository? Do you have a licence? Can people cite your software? Are there releases? Is the documentation accessible? Are you using open development practices, such as discussing issues publicly or reviewing contributions openly? All of these help your software become more trustworthy and more reusable.

The aim here is not to punish for not being perfect. It is about honesty and visibility. If your software is not yet public, you can say so. If you plan to publish it soon, say that. If you rely on open practices internally but haven't released the code, that is also a valid state. The story merely captures the moment in time.

Why FAIR & Open Matter

They increase:

- Trust
- Reuse
- Community uptake
- Research visibility

The benefits of being FAIR and open are well understood within RSQKit, but this is also a chance for you to describe the benefits your own project has experienced. Perhaps external users contributed improvements. Perhaps a public repository increased the visibility of your research. Perhaps a licence helped collaborators use your software safely. Whatever the case, this is where you explain how openness has supported your work.

Section 8: Documentation

How users and developers learn to work with your software.

Documentation is often the section people worry about most, because it feels like an admission of guilt. But the template handles this gently. It asks only for a short subtitle and a description of how the documentation is organised, maintained and accessed.

What To Describe

- Where documentation lives
- How it's maintained
- What forms it takes (README, wiki, notebooks, tutorials)
- You do not need perfect documentation - you need *honest* documentation.

You do not need to claim perfection. You can state plainly that your documentation is minimal, or scattered, or still being developed. You can explain that the README is the primary entry point, or that you maintain a wiki, or that you provide hands-on examples. Documenting the documentation helps others understand how to begin, and it helps you understand where the project might benefit from further investment.

Section 9: Sustainability

How the software is managed, governed and supported over time.

What Goes Here

- Governance structures (formal or informal)
- Funding
- Project activity levels
- Maintenance responsibilities
- Plans beyond current grants

This is the section that most research projects avoid thinking about. Sustainability feels like something we will deal with "later". But the story treats it as part of the natural lifecycle. You are simply describing what exists today: governance structures, maintenance plans, funding sources, project activity and intentions beyond the grant period.

If your project has governance - even informal governance - you write it down. If you have regular meetings, or shared responsibilities, or decision-making flows, include them. If you don't have any governance at all, you can say that honestly. Many projects are sustained by one or two people working at the edge of their time, and that is still worth documenting.

Funding is always important to mention. Perhaps your project is grant-funded. Perhaps it is maintained by a lab technician. Perhaps it is entirely volunteer effort. These are all real situations that shape the sustainability of the work. Writing them down also helps managers and funders understand where support is needed.

Finally, describing the level of project activity - whether slow, stable, or very active - gives readers a sense of lifecycle. Not everything needs continuous changes. Some tools are mature and stable. Others are rapidly evolving. A story captures that so others don't make incorrect assumptions.

Section 10: References

A simple list of cited materials: documentation, tools, practices, influences.

The last section, "References", is straightforward: it invites you to cite any sources, inspirations or documentation that influenced the story. This is particularly useful if you have derived practices from other communities, or if your software is part of a larger ecosystem. It shows your project's place in a wider research environment.

What This Template Really Does

It surfaces context you *already know* but may not haven't written down - the context that collaborators need.

Bringing it together

When you read through the template as a whole, you can see that it forms a complete picture of your project - the problem, the people, the technology, the practices, the tools, the openness, the documentation, the sustainability. No section is complicated. None requires long text. But together, they give a rich and honest view of your software as it exists today.

The Aim Today

Not to finish a story. But to *begin* - to let the template ask you the questions that strengthen your research.

- Support Materials https://tinyurl.com/EVERSE202511-RSS

This walkthrough is not meant to tell you what your story should look like. It is meant to give you confidence that you can write one. The template is simply a set of invitations - prompts that help you surface context that would otherwise remain hidden. And once you have that context written down, everything else in your research becomes easier: collaboration, onboarding, reproducibility, planning, grant writing, peer review and long-term stewardship.

The important thing is that you start. You don't need to finish during this seminar. You don't need perfect answers. You just need to begin asking the questions that the template encourages. That is enough to strengthen your research and make your software more visible and more resilient.