Strengthen your Research by Telling Its Software Story

2 - What Are Research Software Stories?

EVERSE Network Training, Michael Sparks (Nov 2025)

Agenda

First Part

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

Second Part

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

Moving on

What Are Research Software Stories?

A short, grounded explanation of what these stories actually are - and why they take the shape they do.

Now that we've set the context, let's talk more directly about what a Research Software Story actually is. You've heard the general idea, but for the next ten minutes I want to give you a more concrete, grounded sense of what these stories actually look like, why they are structured the way they are, and what makes them useful in practice - not just in theory.

Briefly, I'll show you Baler's from RSQKit. https://everse.software/RSQKit/baler_research_software_story

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

Not Technical Documentation

- Not installation notes
- Not API references
- Not workflow diagrams
- These assume someone already wants to use the software.

Most of us, when we think about describing our software, jump instinctively to technical documentation: installation notes, dependencies, API references, maybe a workflow diagram if we're feeling ambitious. Those things are important, of course, and they all serve a purpose. But technical documentation is usually aimed at someone who already knows they want to use the software. It doesn't really answer the bigger questions of why the software matters, or how it fits into the research, or what decisions shaped its development. And it certainly doesn't help someone who needs to understand the project at a conceptual level before they work with you.

Stories Fill the Conceptual Gap

A story explains why the software matters, how it fits into the research, and what decisions shaped its development.

- Short, structured
- Easy to read and share
- A blend of abstract + project brief + institutional memory

A Research Software Story fills that gap. It's a short, structured narrative that captures the key elements of a project in a way that is easy to read, easy to share, and easy to refer back to. Think of it as a cross between an abstract, a project brief, and a bit of institutional memory - all rolled into one concise document.

Why Call It a "Story"?

Not to be literary - but to focus on:

- the shape of the project, how it came to be
- the problem it solves
- who depends on it
- what assumptions underpin it
- how it supports research

The term "story" can sound slightly unusual at first, especially in a scientific context where we're used to analysis, structure, and evidence rather than narrative. But the point of calling it a story isn't to make it literary or dramatic. It's to encourage us to step back from the code and ask a more fundamental question: what is the shape of this project? What problem is it trying to solve? Who depends on it? What assumptions does it make? How does it support the research?

Narrative in a Scientific Sense

A clear, jargon-free explanation of the research activity surrounding the software.

These are all narrative questions. Not fictional narrative, but explanatory narrative - the narrative of a research project, expressed clearly and without jargon.

Shared Structure Matters

- RSQKit uses one template
- Cross-domain, cross-disciplinary
- Works for physics, biology, humanities, climate science
- Same core questions for any research software project

Now, one of the reasons Research Software Stories work so well is that they have a very consistent structure. They're built around a template that RSQKit uses - a template that was deliberately designed to work across scientific domains. That means it's not tailored for physics or biology or climate science or humanities computing. It's deliberately generic so that the same kind of questions can be asked of any project, regardless of the domain or scale.

Key Elements of a Story

We'll now briefly introduce each element:

- Problem
- Community
- Technical Aspects
- Software Practices
- Tools
- FAIR and Open
- Documentation
- Sustainability

So what are the key elements of a story? Let me walk you through them at a high level.

Problem

Clarifies the scientific or technical challenge the software addresses.

- Problems change over time
- Assumptions drift
- Today's work may differ from original aims
- Surfacing this is illuminating

The first section is the Problem. This is where you articulate the scientific or technical challenge that your software addresses. Every project begins with a problem, but we rarely write it down clearly. Over time, it becomes implicit, embedded in the code or the workflow. This section brings it back to the surface. It's surprisingly revealing: sometimes the problem today is not exactly the same as the problem the software was originally built to solve. That gap is important to recognise.

Community

Who builds the software and who uses it:

- Developers
- Contributors
- Internal/external users
- Distinct groups and roles

Next is the Community - the people who develop the software and the people who use it. This isn't just a list of names. It's about understanding the relationships and the roles. Is it built by a single PI and one PhD student? Is it maintained by an RSE team? Does it have external contributors? Are there distinct user groups? Who relies on the outputs? This part is crucial because software is fundamentally a social activity. Most problems in software are not technical; they're organisational or communicative. Understanding who is involved - and who is missing - is often half the battle.

Technical Aspects

Core technologies and constraints:

- languages, frameworks
- operating environments
- data models
- dependencies

Not a full inventory - just the essentials.

Then we have the Technical Aspects. This is where you describe the core technologies: languages, frameworks, dependencies, operating environments, data models, and so on. It's not meant to be exhaustive. You're not listing every version number or every library. Instead, you highlight the things that define how the software works and what constraints it lives under. This is usually the shortest section, and yet it provides essential context. It tells readers what kind of ecosystem the project sits in.:::

Software Practices

Captures the processes that keep the project alive:

- version control
- review processes
- decision-making
- communication patterns
- testing (formal or informal)

Makes invisible work visible; Reveals gaps constructively.; Supports more sustainable workflow.

After that comes Software Practices. This is where you capture the processes that keep the project alive. How do you manage the code? How do you review changes? Do you test anything? How do you make decisions? How do you communicate? Many research groups do far more than they think here - but because it's informal or habitual, it doesn't get written down. This section helps make that invisible work visible. And if there are gaps - and almost all projects have gaps - it helps you notice them in a constructive way.

Tools

Key infrastructure:

- version control platforms
- workflow managers
- CI systems

- containers, clusters, cloud, scripts

Not exhaustive - just the essentials.

The next section covers Tools - the infrastructure you rely on. Version control platforms, workflow managers, CI services, container images, build systems, cluster environments, shared drives, cloud platforms, local scripts. You don't need to list everything, but naming the key tools helps others understand not just how the software is built, but where it lives and how it moves through the research environment.

FAIR and Open

- Findable
- Accessible
- Interoperable
- Reusable

A chance to reflect on openness, licences, releases, and data formats.

Then comes FAIR and Open. FAIR - Findable, Accessible, Interoperable, Reusable - is increasingly central to modern research practice, and many software projects already align with FAIR principles without realising it. This section is an opportunity to make that explicit. Is the code public? Does it have a licence? Are releases versioned? Are data formats standard? Do external users know how to access it? Again, you don't need to be perfect; you just need to be honest about where you are.

Documentation

Where users learn more:

- README
- wiki
- handbook
- examples/notebooks
- videos

Focus: what exists, what's missing, where it lives.

Next is Documentation - where users and developers can learn more. This might be a README, a wiki, a handbook, examples, notebooks, or even a video. The aim here isn't to write documentation but to describe what exists and what doesn't. It's often surprising how much documentation a project has scattered across different places, and how useful it is to gather that knowledge in one spot.

Sustainability

Funding, governance, maintenance, future prospects.

And lastly, Sustainability. This covers funding, governance, maintenance plans, and long-term prospects. This is often the section people dread, because very few research software projects have formal sustainability strategies. That's normal. The point here isn't to force you into one. It's to help you reflect on whether you have the capacity to keep the software alive, what risks you face, and what options you might have for the future.

What the Whole Story Becomes

Not a technical manual. Not a publication. Not a grant proposal. But a clear, honest description of the project *as it is*.

So when you put all these sections together, what you get is not a technical document, not a grant proposal, and not a publication. You get a clear, honest, structured description of the project as it actually exists - the project as lived, not idealised.

Why Short Matters

Stories are deliberately:

- short
- factual
- accessible
- easy for any domain to read

A Research Software Story is short. It's factual. It's accessible. It's grounded. And because it uses a common structure, it's easy for people in other fields to read. Someone in computational biology can learn from a story written by someone in particle physics. Someone in environmental modelling can learn from someone in digital humanities. The structure creates a shared language - not for code, but for practice.

Stories Create a Shared Language

A shared structure for capturing practice - not code.

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

That's what Research Software Stories are. They're a way of capturing the practical reality of software development in research: the decisions, the relationships, the assumptions, and the contexts that shape the work.

In the next section, we'll talk about why it's worth writing one - and why the benefits can be larger than the effort involved.