Strengthen your Research by Telling Its Software Story

3 - Why Write Research Software Stories?

EVERSE Network Training, Michael Sparks (Nov 2025)

Agenda

First Part

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

Second Part

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

Moving on

Why Write Research Software Stories?

Research Software Stories matter because they help the real people who depend on, join, support, fix, or evaluate your project.

We've talked about what a Research Software Story actually is, and we've explored the template that structures it. Now I want to turn to a more fundamental question: why write one at all?

And to answer that, I always start with the same point: Who is this for? Because if we can identify who benefits, everything else follows very naturally. Once you've named the audience, the what, why, when and how fall into place almost automatically.

So rather than thinking about software in the abstract, I want to talk about the real people who interact with your project - the groups we all know from day-to-day research life. And I want to name them explicitly, because when you actually list the people who depend on your software, or who will one day need to understand it, the rationale for writing a Research Software Story becomes almost self-evident.

Five Audiences

We focus on five concrete groups:

1. Researchers
2. Early Career Researchers
3. Research Software Engineers
4. Managers and Coordinators
5. Funders, reviewers, evaluators

The audiences I'm going to talk about are:

Researchers - the domain experts whose science depends on the software.

Early Career Researchers - the students and postdocs who inherit and extend that work.

Research Software Engineers - the specialists who build, debug, improve and stabilise research software.

Managers and coordinators - the people who shape project direction, resources and planning.

Funders, reviewers and evaluators - the people who need to assess credibility, plausibility and impact.

Why Start With "Who"?

When you start with the people involved, the what, why, when, and how follow naturally.

These are the real "who". Let's take them one at a time, and walk through what they need, why they struggle, when a story is most useful, and how a story actually helps them.

1. Researchers

Researchers rely on the software to carry out the research itself.

Our first group is Researchers - the people whose research depends on the software

Researchers - the scientists, scholars and analysts using the software - are always the core audience. Their priority is not the code itself, but the research the code makes possible. They want to produce trustworthy, interpretable results. They want to do that efficiently. And they want to do it without everything falling apart after a few months.

Loss of Context

Teaching, travel, meetings, supervision, writing - all break continuity. Returning to old workflows becomes challenging.

- Assumptions can be forgotten over time
- Context becomes harder to recover

Researchers can struggle with something very common: loss of context. Research environments are intense, and people jump between teaching, writing, travel, conferences, supervision and their actual research. It's very easy to forget what a particular script assumed, or why a workflow was designed the way it was. And when they return after a gap, they spend a huge amount of time trying to reconstruct their own mental model.

Reconstructing Mental Models

Researchers often try to rebuild their own past thinking:

- why a workflow was structured
- what assumptions were made
- why certain decisions mattered

They also struggle with explaining their workflows to colleagues. Someone new joins the project, and suddenly they have to articulate why certain steps exist, or why the data is handled in a particular way. When that explanation is incomplete, onboarding becomes frustrating for everyone. And reviewers, especially critical ones, will often ask questions about assumptions and design choices that researchers didn't realise needed writing down.

Explaining to Others Is Hard

- New team members need explanations researchers don't always have written down.
- Partial explanations slow down everyone.

This is where a Research Software Story helps them enormously. It gives researchers a structured way to explain the project to new collaborators. Instead of diving into folder structures or notebooks straight away, they start with the higher-level picture: what the software is for, what problems it solves, and who uses it. It becomes a memoryf aid, because the act of writing it forces them to articulate things they would otherwise forget. And it becomes a reference point when they are writing methods sections or grant applications - instead of having to re-derive the rationale for the software, they can draw directly from their own story.

How Stories Can Practically Help Researchers

- Provide a structured overview
- Support onboarding
- Support grant writing

This is where a Research Software Story can help them enormously. It gives researchers a structured way to explain the project to new collaborators. Instead of diving into folder structures or notebooks straight away, they start with the higher-level picture: what the software is for, what problems it solves, and who uses it. It becomes a memory aid, because the act of writing it forces them to articulate things they would otherwise forget. And it becomes a reference point when they are writing methods sections or grant applications - instead of having to re-derive the rationale for the software, they can draw directly from their own story.

When Researchers Use Stories

Stories are how we give people context and history.

Whether formally or informally captured, researchers do use stories.

- Beginning a grant proposal
- Onboarding a new student or postdoc
- Discussing work with an RSE team
- Reproducing older work

A common format can help

Researchers tend to rely on stories (formal or informally captured) at key moments. When they're preparing a grant proposal, a good story gives them half of the "software justification" section immediately. When they're onboarding a new postdoc, the story gives them something to hand over in the first meeting. When they're talking to an RSE team, the story gives the RSE exact context and constraints, saving days of discussion. And when they're reproducing old results, the story cushions them from the pain of trying to reconstruct their own thought-process months later.

Ultimately, a story helps researchers capture the purpose, assumptions, dependencies and context that the code itself can't show. That is what keeps their science reproducible, understandable and defensible.

2. Early Career Researchers

ECRs inherit codebases and need to make sense of them quickly.

Next let's consider Early Career Researchers - the people joining existing work

Early Career Researchers - students and postdocs - are usually the group most affected by the absence of context. They are bright, enthusiastic, and ready to contribute, but they are often parachuted into software ecosystems that have grown organically, chaotically, or under pressure. Their challenge is not lack of ability, but lack of orientation.

ECR Challenges

- Dropped into complex systems
- Limited time to understand context
- Often expected to be productive quickly

They are usually given a large codebase, a few folders of scripts, and some vague verbal explanation. They are expected to produce results quickly, but no one has articulated the bigger picture for them: how the software fits into the research questions, who else uses it, what assumptions are baked in, or what would break if they make changes in a certain direction.

Missing Orientation

They rarely get a high-level explanation of:

- the research goals
- the scope of the software
- what can be changed safely

They need a starting point that explains the project at a conceptual level. They need to know what the software is actually for before they dive into reading the code.

They need to understand the scope of the project, the conventions the group relies on and the long-term research goals. They also need to know who to ask for help, and which aspects are considered “safe to change” versus “core to the project”.

Stories Give ECRs a Starting Point

A story can provide conceptual orientation before diving into code.

A Research Software Story gives them precisely the overview they lack. It bridges the gap between being new and being productive. It helps them avoid spending weeks reverse-engineering code to understand intent. It tells them which aspects of the software matter scientifically, which helps them avoid making unintentional breaking changes. And it gives them the confidence to operate independently because they understand not just the code, but the purpose behind it.

Supporting Transitions

Stories help ECRs when:

- joining a project
- returning after breaks
- switching between related tools
- writing theses

This matters the most at the start of a project, when they are trying to find their footing. It matters when they inherit legacy code that may not be well documented. It matters when they are switching between related subprojects and need to reframe the mental model quickly. And it matters again when they are writing their thesis, because they need to articulate how the software supports their research.

Skill Growth

Stories support progression:

- starting → competent → confident → expert

Thinking in terms of skill levels, a story helps ECRs progress. When they are starting, the story gives them orientation. When they become competent, the story helps them navigate the project. As they gain confidence, the story helps them identify areas where improvements matter. And for the few who reach expert level, the story becomes a foundation for redesigning or extending the project safely.

3. Research Software Engineers

RSEs join when things are unclear, fragile, or need improvement.

Moving on - Research Software Engineers - the specialists who build, fix and support research software

Research Software Engineers often arrive at the point when things are fragile or unclear. They are asked to “help make this work”, often without context. They need to understand the scientific intent behind the software, the constraints under which it was built, the workflows it supports, the data it handles, the assumptions that shaped it, and the communities that depend on it.

RSEs Need the Conceptual Model

Before fixing anything, they must understand:

- scientific intent
- constraints
- assumptions
- data
- workflows

An RSE’s first task is always the same: construct the conceptual model. Until they understand what the software is for, and how it fits into the research, they cannot design solutions that are safe or sustainable. Without that conceptual model, they spend a huge amount of time guessing which parts of the code are critical and which parts are incidental.

Guesswork Is Expensive

Without context, RSEs may spend significant tim reconstructing intent.

A Research Software Story saves them that guesswork. It answers the questions RSEs usually spend days or weeks reconstructing. It explains what the software is meant to do, how it supports the science, what constraints matter and what compromises were intentional. It tells them what success looks like from the research point of view, allowing them to propose solutions aligned with actual scientific goals.

Stories Support RSE Work

They help when:

- joining mid-stream
- debugging unfamiliar code
- estimating feature work
- planning refactoring

Stories can help RSEs when joining projects mid-stream, when debugging code they have never seen before, when estimating how long it will take to implement new features, or when planning major refactoring. It also helps when advising on workflows, because they can see where containers, CI pipelines or automated tests would make the biggest difference. And it helps when writing sustainability plans for grants, because they understand the context as well as the implementation.

Supporting RSE Skill Levels

Stories can help:

- starting RSEs learn context
- competent RSEs identify missing practices
- confident RSEs propose improvements
- expert RSEs refine architecture with respect to research goals

Looking again at skill levels: starting-level RSEs learn scientific context quickly from the story. Competent RSEs use the story to identify missing practices. Confident RSEs propose improvements more effectively. And expert RSEs use the story to align architecture with long-term research vision.

RSEs and RSQKit

Stories reveal which RSQKit tasks are relevant - documentation, workflows, testing, licensing.

The connection to RSQKit is especially strong here. Stories naturally reveal where the project needs specific RSQKit tasks - whether related to documentation, workflows, licensing, testing or sustainability. The story identifies the gap; RSQKit tells them how to address it.

4. Managers & Coordinators

Managers ensure coherence, planning, staffing, governance and reporting.

Our next group is Managers and Coordinators - the people ensuring coherence and direction

Managers and coordinators are often the glue holding research projects together, even if their work is not visible in the code. They manage resources, plan timelines, coordinate staff, handle reporting obligations, and ensure that projects remain aligned with broader goals. But to do any of that effectively, they need clarity about what the software does, who depends on it, and how risky or fragile it is.

Managers Need Clarity Too

They need to understand:

- who depends on the software
- how sustainable it is
- what risks exist
- where bottlenecks appear

They need to understand the broader ecosystem around the software: which teams rely on it, how onboarding works, whether the project is sustainable, and where

the bottlenecks are likely to appear. They are responsible for anticipating risks before they cause disruption, which is almost impossible when software knowledge is locked inside individual heads.

Stories Support Coordination

They help with:

- planning meetings
- deciding priorities
- allocating RSE time
- understanding project evolution

A Research Software Story gives them a single, authoritative overview. It becomes the reference point they use when planning meetings, coordinating interdisciplinary teams, negotiating priorities, or deciding where to allocate RSE time. It helps them identify staffing needs and training needs. And it gives them a way to track how the project evolves over time without needing to understand every technical detail.

Stories can assist during transitions

- new team members
- departures
- new funding stages

Managers rely on stories during annual reviews, institutional reporting, consortia reporting, staffing allocation planning, and during any transition between project phases. Whenever a team member leaves, or a new team member joins, or the project enters a new funding stage, a story increases stability and reduces uncertainty.

5. Funders, Reviewers & Evaluators

They rarely use the software but decide whether it gets funded.

Our final group is less obvious, perhaps because their job is often to decide whether work is worth funding or doing.

This final group rarely interacts with the software directly, but their influence can determine whether a project lives or dies. They need to understand the purpose of the software, the community it serves, the scientific value it enables, and the sustainability of the work. They are not looking at the code; they are looking at credibility.

What Funders Need

Understanding of:

- purpose
- users
- scientific value
- sustainability

Funders need to know that the software is necessary, that the community actually needs it, and that the project is capable of delivering the intended outcomes. Reviewers want to know that the software is aligned with the research aims, that the assumptions are reasonable, and that the project is not overreaching. Evaluators want clarity about sustainability, risk, community uptake and future needs.

Credibility Matters

A story explains community need, rationale, assumptions and relevance.

A Research Software Story speaks directly to all those concerns. It provides a narrative explaining what the software does, who relies on it and why. It describes the scientific relevance of the work and the rationale behind key decisions. It outlines sustainability plans - even if minimal - and identifies risks openly. This is exactly the information evaluators need in order to assess feasibility, reliability and impact.

How Stories Help Evaluators

They shape:

- feasibility
- reliability
- impact
- resource allocation

Funders rely on stories during applications, renewals, mid-term reviews and final reports. Reviewers rely on them when judging plausibility. Panels rely on them to allocate resources. And because stories connect naturally to RSQKit tasks, they also serve as evidence that the project is taking responsible steps toward quality and sustainability.

Cross-Cutting Benefits

Stories reduce friction across the whole project.

Cross-cutting benefits - reducing friction, guiding improvement, securing reproducibility

Friction Slows Research

Time is lost to:

- onboarding
- misunderstood assumptions
- reconstructing context
- repeated explanations

So far I've talked about each audience separately, but there are broader systemic benefits as well. The biggest is friction reduction. In research, friction kills productivity more reliably than bugs do. We lose time onboarding, explaining context, reconstructing old workflows, miscommunicating assumptions, repeating work unnecessarily and guessing intent behind code.

Stories Reduce That Friction

They become a central, reusable reference for everyone.

A Research Software Story reduces that friction across the entire project. It becomes a centralised, structured, reusable resource that everyone can refer to. It doesn't need to be long - it just needs to be clear. And because it aligns purpose, assumptions and workflows, it naturally increases reproducibility. It ensures that methods can be defended. It helps teams rediscover their own reasoning months later.

Stories Guide Improvement

- They can reveal missing documentation, fragile dependencies, unclear workflows, ambiguous ownership.
- Stories can highlight necessary RSQKit tasks and improvement pathways.
  - To yourself and your team
  - To others seeking to emulate your success

Stories also guide improvement. They make gaps visible. They reveal missing documentation, unclear workflows, fragile dependencies or ambiguous ownership. They can highlight which RSQKit tasks are relevant, giving teams a natural improvement pathway without imposing heavy process. And stories help future-proof the project by capturing the "why" behind decisions - the piece most vulnerable to loss when people move on.

If shared with the RSQKit team, it can also helps others to follow in your footsteps.

So why write Research Software Stories?

- Stories preserve the "why" behind decisions - protecting against context loss.
- Stories help everyone: users, newcomers, RSEs, supporters, evaluators.

So why write Research Software Stories?

Because they help the people who use your software, the people who join your project, the people who fix your project, the people who support your project, and

the people who fund your project. They reduce friction. They improve clarity. They strengthen collaboration. They make your research more reproducible, more trustworthy and more resilient.

They make the impact of your software visible. They help you see your project clearly. They help you explain it confidently. And they help you develop it sustainably.

The Core Message

Stories strengthen research by making implicit knowledge explicit.

… so let's look at what that means in detail.

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

Ultimately, they strengthen your research by telling the story that already exists - often only people's heads. So let's now move onto how to *write* a story - walking through the template itself.