Level 3: Advanced | All Librarians

Module 16

Vibe coding for librarians

The first practitioner focused vibe coding curriculum for librarians. No programming required. You will describe what you want in plain language and watch it become a working tool. We will build real library tools together, and we will also reckon honestly with the risks.

By Yulia Brusova

20 min read · Reviewed June 2026

AI for Academic Libraries · ai-in-academic-libraries.vercel.app

WHAT YOU'LL BE ABLE TO DO

1. Build a simple functional tool using plain language prompts to an AI coding tool
2. Describe the vibe coding workflow and how it differs from traditional software development
3. Identify three library workflow problems that could be addressed with a custom built tool
4. Evaluate when building a tool is the right approach versus configuring or purchasing an existing one
5. Articulate the five primary risks of vibe coded tools and how to mitigate each in a library context
6. Apply a documentation standard to a tool you have built, including what it does, what data it touches, and what it cannot do
7. Identify equity considerations in making AI coding tools available to library staff
8. Assess a proposed vibe coded tool for appropriateness relative to your institution's patron privacy and security obligations

> *In February 2025, Andrej Karpathy, one of the foundational researchers behind modern artificial intelligence and former director of AI at Tesla, introduced a term that has since entered the vocabulary of technology practitioners worldwide. Vibe coding, as he defined it, is a mode of software development in which one describes a desired tool in plain language and allows an AI system to write the underlying code. His framing was deliberately informal: 'I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.' What he did not say, and what this module takes seriously, is that a librarian with no programming background can use exactly this workflow. This module shows you how to do it, and shows you what to watch for when you do.*

What vibe coding is, and why it is a library problem

Vibe coding is building software by describing what one wants in plain language and allowing an AI to write the underlying code. There is no programming involved on the part of the person doing the building. One describes the tool, reviews what the AI produces, describes

what should change, and iterates until the result is functional. The process is conversational rather than technical.

This is a genuine shift. For most of the history of software development, building a custom tool required either programming knowledge or a budget sufficient to hire someone who had it. Academic libraries have faced a specific version of this constraint for decades: tool needs that are too particular for a vendor product, too small to justify a development contract, and too complex to solve without code. The result has been a persistent accumulation of workarounds: spreadsheets managing workflows that deserve proper tools, paper processes persisting because no one could build the replacement, staff time spent on tasks that could be automated if automation were accessible.

Vibe coding closes some of that gap. It is worth naming honestly why this capability is appealing in a library context, because understanding the motivation is part of using the capability responsibly. In a May 2026 article in C&RL News, Ava Wallace, an LIS student writing alongside a library director and a senior leader in the profession, describes AI tools generally as measures that address a chronic funding crisis, useful precisely because libraries have been asked to do more with less for so long, but not a substitute for the sustainable investment communities actually need. That framing applies directly to vibe coding. It is a workaround made newly accessible, and naming it as such is more honest than describing it as a solution.

For example, the following tools are now within reach of a librarian with no programming background:

- A citation scavenger hunt tool for library instruction sessions
- A database recommendation quiz that matches patrons to the right resource based on three questions
- A research log template that students complete as they work through a project
- A simple intake form that routes patron requests to the appropriate librarian
- An interactive checklist for evaluating sources
- A subject specific glossary for patrons in a technical field
- A quiz that tests whether students can identify a hallucinated citation

None of these require programming knowledge. They require the ability to describe what one wants clearly enough that an AI system can build it.

There is one constraint worth naming directly at the outset: vibe coded tools are functional, not polished enterprise software. They work. They are reasonable in appearance. For patron

facing tools that must be sustained over multiple years, vibe coding is best understood as a rapid prototyping approach: a way to test whether a tool is worth building properly. For internal staff tools, it frequently serves as the final product.

The tools to know

Lovable (lovable.dev) is the most beginner accessible vibe coding tool currently available. One describes an application in a chat interface, Lovable builds it, and a live preview appears immediately. The tool is designed to produce web applications that look polished without requiring the builder to make design decisions. It is the recommended starting point for most library vibe coding projects.

Strengths: Strong default design, straightforward iteration, live preview. Well suited to patron facing tools where visual quality matters. Limitations: Less control over structural decisions; complex logic can be difficult to implement precisely. Pricing: Free tier has limited builds. Paid tier ($20/month) is appropriate for active use.

Replit (replit.com) is a development environment with strong AI assistance built in. It is more flexible than Lovable and supports a wider range of project types. The AI within Replit, called Agent, can build more complex tools and allows the builder to see and interact with the code directly if they choose.

Strengths: Very flexible, handles complex requirements, supports more unusual tool configurations. Appropriate for tools that need capabilities Lovable cannot provide. Limitations: Defaults are less visually polished; requires slightly more comfort with iteration. Pricing: Free tier is functional for most library tool builds.

Claude (claude.ai) or ChatGPT directly: One describes a tool in plain language and asks the AI to write the code. The AI produces code that is then pasted into a text file with an .html extension and opened in a browser. This approach works for simple self contained tools such as calculators, quizzes, and interactive pages.

Strengths: Free, immediate, works well for simple tools. Reading the code, even briefly, builds some understanding of what the tool is doing. Limitations: Iteration is manual; each revision requires repeating the process. Requires knowing how to open an HTML file in a browser.

The recommendation for starting is Lovable for a first project. For subsequent projects that require more flexibility or complexity, Replit is the natural next step. Direct use of Claude or ChatGPT is appropriate for quick, simple tools where the overhead of a dedicated platform is not warranted.

The vibe coding workflow: step by step

The workflow is consistent regardless of which tool one uses. The quality of the result depends more on the quality of the description than on any other factor.

Step 1: Write a clear description of what you want. Technical terms are not necessary. One should describe the tool as one would explain it to a colleague who could build anything. A useful description includes: what the tool does and why, who will use it, what inputs the user provides, what the tool shows or outputs, and roughly how it should look.

An example of an insufficient description: "Make me a library app."

An example of a description that works: "Build a web page with a simple quiz. It asks three multiple choice questions that test whether a student can identify a hallucinated citation. Each question shows a citation and asks: Real or Hallucinated? After all three questions, show the score and a brief explanation of each answer. Use a simple, clean design with a navy blue header."

The difference is specificity. The more precisely one describes the inputs, outputs, and purpose, the closer the first version will be to what is actually needed.

Step 2: Let the AI build a first version. In Lovable or Replit, paste the description and allow the AI to generate. In Claude, ask it to write the HTML and JavaScript, then copy the result into a file with an .html extension.

Step 3: Review and iterate. The first version will not be exactly right. That is expected. The next step is to describe what should change: "The colors are wrong. Use dark green instead of blue." "The explanation after each question is not appearing." "Add a Start Over button at the end." "The text is too small on a phone screen."

Step 4: Test as a user. Move through the tool as a patron or student would. Identify what does not work as intended. Describe each problem to the AI. Repeat.

Step 5: Share or deploy. Lovable and Replit both provide shareable links that work immediately. For HTML files built directly through Claude or ChatGPT, the file can be uploaded to a library website or a simple file hosting service.

The iteration cycle (describe, build, review, revise) is the core skill. It is conversational in nature, and it rewards clarity and specificity at every stage.

Real library tools built this way

The following examples represent tools a librarian with no programming background can build in a single afternoon or evening. They are illustrative of the category of problem vibe coding addresses well.

Citation Reality Check is an instruction tool presenting students with five citations, some genuine and some hallucinated by an AI, and asking them to identify which is which. Immediate feedback is provided after each selection. The tool works in any browser, requires no login, and can be shared as a link before a library instruction session or embedded in a LibGuide.

Approximate build time: 45 minutes including iteration.

Database Matchmaker is a reference tool that asks three questions (subject area, type of information needed, and level of depth) and recommends the library's databases based on the answers. The builder provides the recommendation logic; the AI builds the interface.

Approximate build time: 90 minutes including writing the recommendation logic.

Research Log is an instruction support tool presenting students with a structured form they complete as they work: their research question, search terms they tried, databases they used, sources they found useful, and questions that arose. The form allows students to export a summary of their process. It is useful for scaffolding research as a skill rather than an outcome.

Approximate build time: 60 minutes.

Subject Glossary is a patron support tool providing an interactive glossary for a specific subject area (nursing, legal studies, social work, environmental science) in which patrons can search terms and receive plain language definitions. The builder provides the term list; the AI builds the searchable interface.

Approximate build time: 30 minutes once the term list is prepared.

These are not complex applications. They are tools that would previously have required either a developer, a vendor product that approximated the need, or simply going without. The shift in what is accessible in a single afternoon is the point.

When to build vs. when to configure

Vibe coding is not always the right answer. Before building anything, it is worth asking a series of questions that will clarify whether building is the appropriate response.

Does a tool for this already exist? LibCal handles appointment scheduling. LibGuides handles resource organization. Most integrated library systems and discovery layers have features that are underused. Configuring an existing tool often solves a problem faster and more sustainably than building a new one. Before building, investigate what is already available.

Does it need to integrate with another system? If the tool must connect to an ILS, Alma, a database, or another institutional system, vibe coding can produce a starting point but will frequently encounter its limits quickly. Integration typically requires API access and more than vibe coding alone can provide.

Does it need to be maintained over the long term? Vibe coded tools are easy to build and can be fragile to maintain, particularly when the person who built them is no longer available. For tools that need to function reliably over years, a formal development project or a vendor product may be the more appropriate answer, even if it is slower.

Is it patron facing and high stakes? A broken patron facing tool erodes trust. Any tool deployed to patrons should be tested extensively and should have a plan for when something goes wrong, including when the person who built it cannot diagnose the problem.

When building is the right choice:

- Prototyping an idea before committing to a formal build
- Internal tools used only by staff
- One time instruction tools such as quizzes, games, or demonstrations
- Tools so specific to a local context that no vendor product could address them
- Any situation where the iteration speed of vibe coding is itself a meaningful advantage

The risks you must take seriously

The most important characteristic of vibe coding is also its most significant liability: the person who builds the tool does not know what the code is doing. This is not a temporary limitation that will be engineered away. It is structural to the approach. Understanding the risks this creates is a professional obligation, not an optional consideration.

The code opacity problem. When a librarian builds a tool using vibe coding, the resulting code is produced by an AI and may be dozens or hundreds of lines long. The builder almost certainly cannot read it. They cannot tell whether it is handling edge cases correctly, what happens when a data format changes, or what occurs when a user does something unexpected. The tool works until it does not, and when it does not, the builder may have no way to diagnose why. This is acceptable for low stakes internal tools. It is not acceptable for any tool that gates access to resources, stores patron data, or makes decisions with meaningful consequences for patrons or the institution.

Security and patron privacy. Patron data carries professional and often legal protection obligations. A vibe coded tool that is connected to patron records, authentication systems, or personally identifiable information requires review by someone with security expertise before it is deployed. There is no exception to this. An acquisitions reporting tool that processes only institutional financial data carries different risk than a tool that handles student research records or reference requests. The distinction must be made explicitly before any tool goes live.

Maintenance and institutional memory. A vibe coded tool was built through a conversational process with an AI. That conversation is gone. When the person who built the tool leaves the institution, or simply cannot remember what decisions were made during iteration, the tool becomes effectively unmaintainable by a successor who cannot read the code. Libraries have long faced the problem of institutional knowledge residing in individual staff members rather than documentation. Vibe coded tools create a particularly fragile version of that problem.

Skills that may not develop. In a May 2026 article in C&RL News, Trevor A. Dawes articulates a concern about AI generally that applies directly here: when AI can instantly generate seemingly plausible answers, patrons and staff may lose the capacity to evaluate sources, understand what a system is producing, or distinguish synthesis from analysis. The same dynamic operates in vibe coding. A librarian who learns to build tools entirely through AI assistance from the beginning of their career may not develop the data literacy to understand what a query is returning, what a formula is calculating, or what an algorithm is selecting. This is worth examining carefully in any staff development program built around these tools.

Equity across library staff. Research published in C&RL News in May 2026 makes a point that applies directly here: free is not the same as full. The AI tools that make vibe coding effective (Claude Pro, Cursor, GitHub Copilot) are subscription products. The librarians currently using these tools are, in most cases, the librarians who already had the technical curiosity to experiment, the institutional latitude to try new approaches, and access to paid

tools through personal subscriptions or existing institutional licenses. If vibe coding becomes a meaningful capability for library services, treating it as something individual staff must procure on their own creates a two tier workforce. Equitable access to the tools is a prerequisite to equitable distribution of the capability.

Vibe coding and the vendor relationship

There is a consequence of this capability that is directly relevant to the relationships academic libraries have with their technology vendors, and it deserves explicit attention.

The frustration with vendor lock in is a recurring theme in library technology conversations: platforms that do not quite do what a library needs, feature requests that wait years for implementation, interfaces designed for an average use case that does not match a specific institution's workflows. For decades, the practical response to "why can we not simply build what we need ourselves?" was consistent: because building requires developers, developers cost money, and libraries do not have that budget.

Vibe coding changes that calculation. Not entirely; integrated library systems, discovery layers, and electronic resource management platforms are not going to be replaced by vibe coded applications. However, for the category of small, workflow specific tools that libraries have always needed and never been able to build, the barrier has dropped from "requires a developer" to "requires a clear description and an afternoon."

A librarian who can produce a working prototype of a needed tool in three days is in a different position in conversations with a vendor about feature timelines. This does not mean building a workaround is always the right response to a missing vendor feature; it may not be sustainable, and it may create maintenance burdens that exceed the original problem. However, it does mean that the question of where to invest in building versus where to rely on a vendor has become genuinely worth asking in a way it may not have been five years ago.

Libraries can and should think strategically about this. For workflow specific tools that a vendor is unlikely to build, and for which the maintenance burden is manageable, building is increasingly a realistic choice. For tools that need to be integrated with core systems, that must be sustained over many years, or that require security level reliability, vendor relationships and formal development remain the right answer. Making that distinction clearly, rather than defaulting to either "we cannot build anything" or "we can build everything," is part of what it means to approach this capability as a professional.

Building a sustainable practice, not just a single tool

The most useful framing for approaching vibe coding as a library professional comes from Ava Wallace, writing in C&RL News in May 2026: "Literacy does not only mean how to use something. It also means how to think critically about it, how to assess its accuracy, and how to determine when it is or is not an appropriate tool to turn to." That definition (literacy as judgment, not merely adoption) is the correct frame for building a vibe coding practice in a library.

The goal is not staff members who can build anything using AI tools. The goal is staff members who understand what they are building, know what the tool cannot do, recognize when to stop, and have documented what they have made clearly enough that someone else can use and maintain it. That is a library value applied to a new capability, not a technology value imposed from outside the profession.

In order to build a sustainable practice, the following principles are worth establishing at the outset.

Start with low stakes internal tools. Data cleaning, reporting, internal routing, and prototype instruction tools are the appropriate initial experiments. Patron facing systems and anything connected to authentication or core library systems are not appropriate starting points, regardless of how promising the prototype looks.

Find the people who are already doing this. In most libraries, at least one staff member is already building tools this way without calling it vibe coding. Finding those people, giving them time and access to appropriate tools, and learning from what they have already built is more productive than creating a formal program from scratch.

Establish documentation habits before they are needed. When a tool is built, the builder should produce a single page summary at minimum: what the tool does, what data it touches, what it cannot do, what assumptions were made during the build, and who to contact when something breaks. This does not resolve the institutional memory problem entirely, but it reduces the damage when the builder is unavailable.

Treat tool access as an equity question. If vibe coding is going to be a meaningful capability for library services, access to the AI tools that make it possible should not depend on individual staff subscriptions. Such a situation is structurally equivalent to giving some staff access to library databases and not others; the resource should be available equitably, or the capability cannot be deployed equitably.

Engage IT before deploying anything. Security considerations, data handling requirements, and acceptable use policies are far easier to address before a tool is live than after. A disagreement about policy discovered in advance is preferable to one discovered retroactively.

The question for academic libraries is not whether vibe coding belongs in this environment. It is already here. The question is whether libraries will approach it with the same professional intentionality they bring to any new capability: evaluating it critically, deploying it equitably, and sustaining it with the institutional care that patron trust requires. Those habits are already in the profession. Applying them to this new capability is the work.

PRACTITIONER NOTE

A citation evaluation quiz for a library instruction session is a reliable first vibe coding project: describe the purpose, the format of each question, and the feedback students should receive, paste the resulting HTML into a file, and a usable tool typically emerges in twenty minutes. Such a tool can be revised repeatedly, each time by describing the change rather than touching the code. Students engage with it differently than they do with a slideshow: they are doing something rather than watching something. A database recommendation form for the reference desk is a natural second project, typically taking closer to two hours including iteration, because the logic connecting answers to recommendations is more complex than an initial description captures. Neither of these tools could be built without an AI writing the code. The builder can describe precisely what each tool does but cannot read the code that makes it work. That asymmetry is real, and it should be held in mind every time a new idea is evaluated for whether this approach is appropriate or whether it requires something more robust.

KEY TAKEAWAYS

- Vibe coding is the practice of building functional software by describing what is needed in plain language, with no programming knowledge required. It is already in use in academic libraries, often without a name for it.
- The most beginner accessible tools are Lovable, Replit, and direct use of Claude or ChatGPT. Each has different strengths; Lovable is the recommended starting point for most library use cases.

- The core workflow is five steps: write a clear description, let the AI build a first version, review and iterate, test as a user, and deploy or share. The quality of the initial description determines most of the outcome.
- Vibe coded tools carry real professional risks: code opacity, security and privacy liability, maintenance fragility, potential skills gaps, and inequitable access across staff. Each requires explicit attention, not assumption.
- The capability shifts the relationship with vendors. When building a workaround takes days rather than years, the question of where to build versus where to configure versus where to advocate for a vendor feature becomes genuinely worth asking.
- The goal is not tool adoption; it is tool literacy: knowing what a tool does, what it cannot do, when it is the right approach, and how to document and sustain it. That is a library value, not a technology value.

References

APA 7TH EDITION

1. Karpathy, A. [@karpathy]. (2025, February 2). There's a new kind of coding I call "vibe coding" [Post]. X (formerly Twitter). https://x.com/karpathy/status/1886192184808149383
2. Michalak, R., Dawes, T. A., & Wallace, A. (2026, May). Envisioning AI's role in libraries: Perspectives from an LIS student, a library director, and a university librarian. *College & Research Libraries News, 87*(5). https://crln.acrl.org/index.php/crlnews/article/view/27321

   Note: Module text attributes separate passages to "Ava Wallace" and "Trevor A. Dawes" individually. Both passages originate from this co-authored article; in-text attributions should be updated to cite all three authors or use (Michalak et al., 2026).

ACRL AI Competencies covered

Use & Application

Sub-competencies: 4.4, 4.1 · ACRL AI Competencies (2025)