ARTICLE

From Weberian Rationalization to JavaScript Components

Modularism in Academic Library Software

Mark E. Eaton

ABSTRACT

This paper considers modular approaches to building library software and situates these practices within the context of the rationalizing logics of modern programming. It briefly traces modularism through its elaboration in the social sciences, in computer science and ultimately describes it as it is deployed in contemporary academic libraries. Using the methodology of a case study, we consider some of the very tangible and pragmatic benefits of a modular approach, while also touching upon some of the broader implications. We find that the modularism is deeply integrated into modern software practice, and that it can help support the work of academic librarians.

INTRODUCTION

Modularism is an approach to designing products, whether physical things or more ephemeral things like software code. It is a way of thinking about the composition of tools or products. It is a way to conceptualize them in such a way that their design can be broken down into smaller parts, which should (in theory) be easier to build independently and to fit together. It is a phenomenon that has been described in literatures as diverse as sociology, management, and computer science; indeed, when we begin to think modularly, we can begin to see many specific, interesting commonalities between diverse fields.

What is clear at the outset is that, in its present form modularism is ubiquitous and its influence is vast. As a thought experiment to get us started, consider the interoperability on display at a typical hardware store. To solve almost any home repair problem, the hardware store provides access to a range of interoperable products. Parts work together seamlessly, even from different manufacturers, because they are designed in a modular way. Schilling makes a similar point, for similar purposes, using the example of modular Ikea furniture.[1] As a general rule, Ikea does not make bespoke parts for different lines of furniture; rather the same types of part are used and reused in different products, allowing them to be designed and assembled modularly, at a greatly reduced cost. Likewise, Baldwin and Clark use the similar example of standardized sizes for bedding, such as sheets and mattresses.[2] While it is possible for someone to commission a nonstandard-sized mattress, most of us have obvious incentives to use the standard sizes: modularization makes buying sheets, bed frames and blankets much easier, because they fit together by design. When considering these mundane yet expansive examples, it becomes clear that modularism is pervasive across industries and experiences.

Modularism has important consequences for how librarians do their work, in interesting and perhaps unexpected ways. How library software tools are built—whether in-house or externally—

About the Author

Mark E. Eaton (mark.eaton@kbcc.cuny.edu) is Reader Services Librarian, Kingsborough Community College, City University of New York.

Submitted: 17 January 2025. Accepted for Publication: 16 July 2025. Published: 15 September 2025.

https://doi.org/10.5860/ital.v44i3.17357

influences the work of librarians. Tracing a genealogy of a concept like modularism provides an important handle for thinking about the professional work of technically minded librarians and programmers. While concepts like modularism may not be front of mind for librarians, this paper will argue they are a generative part of how things are built with code in libraries. This paper aims to examine the concept of modularism more closely, to shine some light on aspects of library work that may be largely overlooked. It argues that modularism is a foundational concept in modern library software and consequently should receive more attention in library literature. This paper attempts to address this lacuna by examining modularism through a variety of disciplines: the social sciences, computer science, and library practice. It shows the conceptual line that can be traced from the theoretical to practical, impacting academic library workplaces.

The argument here is a historical one, but also a conceptual one. The literature of modularism will be considered in the first half of this paper, then this framework will be applied to the process of building software at the Kingsborough Community College library. The conclusion is that the current software development choices of the Kingsborough library strongly reflect a modular approach.

LITERATURE REVIEW

Modularism is an idea with a long history in the twentieth century, appearing frequently in the academic and technical literature from the 1960s until today. The first explicit mentions of the term found in this research were Simon and Pieper.[3] Writing in 1958, Pieper references already existing modularisms, specifically pointing to examples that were contemporaneous at his time, such as modular architecture and modular furniture.[4] Indeed, while it is possible to find claims of evidence of modularism that predate both Simon and Pieper—such as Buscalioni et al., who find evidence of modularism in antiquity, and Duchscherer and Keister, who find such evidence in early twentieth century home construction—neither of these claims is particularly credible.[5] Nonetheless, as a result, the origins of modularism remain unclear. While neither Simon nor Pieper can be unambiguously credited with coining the concept, their work reflects an important inflection point where it is possible to see the explicit elaboration of a novel, modern modularism that was gaining traction in their circles in the late 50s and early 60s.

Over the past 60 years, the literature on modularism has demonstrated tremendous interdisciplinary breadth. Many authors have pointed out modularity in their respective fields. Parnas and Maynard pioneered modularism in computer science, while Pieper and VanGundy were concerned with modular management.[6] Kozhukhovskaya et al. describe the concept's applicability in logistics.[7] Nevins, Whitney and DiFazio, and Schilling discuss modularism in manufacturing.[8] Similarly, there is a robust discussion of modularism spanning many decades in both academic and trade publications on the automotive industry.[9] The concept has also been deployed in the biological sciences.[10] Lastly, Currie and Sterelny apply modular thinking to philosophical analysis.[11]

This paper takes a limited approach to sociological theory: it uses some well-known concepts as a way to frame the subsequent analyses of modularism in librarianship. The aim is not to make theoretical contributions to the social sciences; rather, the hope is instead to briefly use some theory from outside of library studies to help with a more perspicacious perspective on librarianship.

Modularism can usefully be seen as an extension of Max Weber's concept of "rationalization."[12] Weber's rationalism is a historical movement toward efficiency, systematization, and reason that

he suggests can be seen in the decision-making of individuals. While recent critiques of Weberian-inflected modernism are worth our attention, authors such as Baldwin and Clark have plowed ahead with a Weberian case for modularism in software practices.[13] Baldwin and Clark, as well as Greeven, implicitly situate modularism within a rationalist and capitalist framework, thereby building on Weber's well-known foundations.[14] In this way, they draw an important connection between the work of software development and a rationalized, capitalist worldview. They allow us to draw a more nuanced picture of contemporary modularism and provide us with a starting point to consider the consequences of software modularism in academic libraries.

Following Greeven, and Baldwin and Clark, this paper will suggest that modular software is a rationalizing project. Similar conceptual and organizational processes are at work whether one is modularizing code or modularizing products for the hardware store.[15] As in Weber, these ideas are deeply tied to capitalism and its logics of efficiency. It would be easy to scratch the surface here to see how modularism is contributing to many very common, and widespread, dissatisfactions with capitalism. However, this paper forgoes such analysis and instead follows Baldwin and Clark's narrower path forward to focus on how rationalist modularism is deployed in software. While our analysis obviously does not erase (or overcome) the problems of rationalization and capitalism, it does offer interesting conclusions. It looks at how specific modular methodologies became a part of the repertoire of computer science, and later, of librarianship.

Software Modularism

Modularism first appeared in software with the idea of "information hiding," described by David Parnas in 1972.[16] This idea was foundational for making interoperable components that provide only minimally necessary interfaces to interact with other components. Parnas' work was very important and timely, because, as Davis, Burry, and Burry noted, "by the late 1960's programming had come of age: the essential mechanics of programming had been developed, the speed of computers was increasing exponentially, and more code was being written. Yet despite these advancements, programmers were struggling to produce and maintain collaboratively written code, in part because the GOTO statements would often tangle and become 'spaghetti code'."[17] A new paradigm for organizing code was needed, and Parnas' modularism made great strides toward solving these problems facing programmers in the early 1970s.

In recent times, there have been numerous treatments of modularism in computer science. This is likely because modularism is so easily integrated into computer science concepts. For example, modular computer programming emerged in the 1970s, but was quickly absorbed into other computing ideas, such as object-oriented programming.[18] This happened in large part because modularism was such a useful base on which to build these ideas. New terms were applied, and the field largely moved on, having fully incorporated the insights of modularism. This absorption of the concept into mainstream computer science was perhaps one of the highest compliments that could be paid to the idea. What is clear is that the impact of modularism is undeniable: for example, the full consequences of the developments from the 1970s can be readily seen in later influential programming texts such as Hunt and Thomas' book, *The Pragmatic Programmer*.[19]

What is modularized code? According to Jeff Maynard, who was writing around the same time as Parnas, "modular programming is a system of developing programs as a set of interrelated individual units (called modules) which can later be linked together to form a complete program."[20] Later, Lavy and Rami described modular software design as "splitting up large computation into a collection of small, nearly independent, specialized sub-processes."[21] A

modular approach can divide a program into modules, each responsible for one aspect of the desired functionality. Once they are defined, modules can then be replaced or extended with minimal changes to the surrounding code.[22]

This very important conceptual step forward in computer science has profoundly affected the software industry in indelible ways since the 1970s. Parnas' ideas continue to be expanded upon much later. Indeed, they are also later applied in software at much higher levels of abstraction. For example, Lev Manovich expands on modular principles in his 2013 book about media editing tools called *Software Takes Command*.[23] In this more contemporary treatment, Manovich claims that computer-based media editing tools function in a modular way. Media-editing software not only offers stand-in replacements for analogue processes and effects, but it also builds upon and remixes analog-based ideas to provide entirely new editing capabilities. For Manovich, this process is deeply modular. In Manovich's argument, abstraction and modularism are closely related in the field of media editing software. In the examples he describes, modules are experienced as discrete, innovative tools that help us with our media work.

Modules are also a core element of modern package managers. Tools like apt (Linux), pip (Python), npm (JavaScript), RubyGems (Ruby), and others are all paradigmatically modular, and sit at the center of modern software development.[24] They are particularly salient examples of modular tools that programmers often use on a daily basis.

Moreover, the elaboration of modularist approaches has continued with the development of "microservices" architectures in computing since the mid-2010s. Microservices, starting with the landmark blog post by Lewis and Fowler in 2014, are further systematically described by Newman in 2015 and 2020.[25] These works outline a strategy whereby maintaining codebases is made easier by breaking large applications into small, highly modularized components that communicate with each other through minimal interfaces, like RESTful services over HTTP. Parnas' information hiding principle is essential to such an approach.[26]

Drawing upon some of the same touchstones from the long history of computer science mentioned above, Velepucha and Flores are very positive about a microservices approach: they detail both the advantages and disadvantages; yet fall squarely in favor of using microservices as a modular architecting principle.[27] Guntakandla is similarly positive.[28] Gan and Delimitrou provide supportive evidence for such claims by doing hands-on benchmarking to document the quantifiable advantages of modular code.[29] Others delve further into the details of securing or migrating to a microservices architecture.[30] Finally, there are numerous examples of modularized microservices being deployed in different contexts, such as education software, manufacturing, building management, networking, Internet of Things devices, and geospatial data management.[31]

What is clear is that the simplification of code provided by modules (and their descendants such as microservices) has proven immensely valuable for the field of computer science.[32] Recently, advances have been made in quantifying the impact of modularism, resulting in the benefits of such an approach becoming clearer.[33] As Seydnejad describes, modular architecture allows for more robust and flexible applications.[34] Modularity is at the core of many modern computing paradigms, from objects to closures, and has helped drive many of the advances that modern software has produced.[35] Modules are "a fundamental method of abstracting complicated text-based code to make it more comprehensible."[36] Modularism's longevity as a concept in programming is attributable to its ubiquitous role in shaping how programming tools are constructed. It is clear that this approach is not limited to a particular piece or type of technology.

These ructions in the world of software have had important effects on other professions, such as librarianship. In academic libraries, building with code powers the work of librarians, but librarians are not merely manipulating their own small programs of limited consequence; rather their work is part of a broad, rationalizing impetus toward modularism, which has been felt across many industries. In this way, librarians are heirs to Weber in their library work.

CASE STUDY

As this argument proceeds further downstream, it is clear that modularism affects our library departments. Whether it is through hardware or software, applied modularity is an unavoidable part of contemporary library work. While most people may not have much say as to whether they participate in a Weberian-inflected modernity (indeed, Weber famously refers to modernity as an unavoidable "iron cage"), librarians nonetheless need to build library services with the software tools that they have on hand.[37] For this reason, the literature on modularization is very relevant to libraries and librarians. This section will describe practical benefits of modularism in code at the Kingsborough library. The point here is not to dwell on the details of a specific implementation, but rather to provide a useful, practical example of how modular approaches are deployed in academic libraries.

At the Kingsborough library, the librarians have taken modularist lessons to heart. In building the library website, the Kingsborough librarians used Vue.js (henceforth referred to as Vue) to modularize their front-end code. Vue is a JavaScript framework that was first released in 2014.[38] As a front-end framework, it is a set of tools to structure code, primarily JavaScript, in a way that allows for clean organization, reactivity, and concision. Vue was chosen because its progressive approach supports relatively small scale and specific use cases and because of its modular approach.[39]

One of the most relevant features of Vue for the purposes of this paper is its use of components. Vue is focused on component-based composition. Component-based design is a very useful—and strongly modular—paradigm for code. Vue components move away from web development's traditional separation of concerns in websites into discrete HTML, CSS, and JavaScript files. Instead, Vue components allow the programmer to think about code as modules first: the programmer can create components at whatever scale makes sense to them. Components are modular in that they take inputs in carefully defined ways. The theory behind this is that Vue components should only include code that is relevant to that component. As a result, there is a significant separation of concerns between components, rather than the traditional separation between file types. In the Kingsborough library, the librarians use separate Vue components for the different navigation features of the library website. Vue components offer them an efficient way to abstract away functionality that had previously been (somewhat messily) included in the site's single JavaScript file. In this way, the move to Vue has greatly improved maintainability. Vue components provide the encapsulation that any modularized approach to programming ideally should have.[40] This is important, because as Seydnejad says, the separation of concerns is one of the goals of modular programming.[41]

There are of course other examples of implementations of modular library-focused software elsewhere in the literature of librarianship. Bradley offers a model for encouraging modular software maintainability, using another JavaScript framework called Svelte.[42] Bradley is also doing front-end development in an academic library. Many of the lessons that he has learned with Svelte have been abundantly clear to the Kingsborough librarians in their work as well. As he says,

"modularization is common in many programming ecosystems and may not seem like much, but I strongly recommend this practice to anyone building apps, especially in JavaScript. This allows for easy debugging of your functions and makes them reusable not only within the app, but within other projects as well."[43] Hahn and Ryckman, who are also working in libraries, have similar insights that they refined while deploying modular approaches in mobile apps.[44]

Moreover, commercially-developed library software often shows evidence of modularism. The Kinsgsborough library uses Alma as its library management system, which exhibits characteristics of a modular, object-oriented approach. Alma's system of APIs for scripting library work is very modular, by offering simplified, mostly unchanging interfaces to what is likely more complex, hidden software.[45] Lastly, LibGuides CMS, another popular tool at Kingsborough library, is modular in both its user interface and its code: it uses a modular system of "boxes" which can be interchangeably used throughout the CMS to obviate the need to rewrite code or content in multiple locations.[46]

The benefits to such an approach are fairly clear. As Schilling says: "when products are made more modular, it enables the entire production system to be made more modular."[47] Moreover, "this kind of architectural approach leads to huge advantages in many aspects of our application fundamentals such as code usability, maintainability, testability, and many more."[48] Organizing code into modules ensures programs are human readable and keeps potential problems to a minimum.

In the end, Vue has vastly improved the maintainability of the Kingsborough library webpage, as compared to the previous vanilla (in other words, with no framework) JavaScript approach. Other complimentary features of Vue, such as reactivity and syntactical conciseness also helped make the code much more readable. But much of the benefit is clearly thanks to modularism: it has made the code more understandable and maintainable. While there are other benefits to Vue that have not been touched upon here, the unambiguous conclusion to draw is that Vue's modularization has had a direct positive impact on the librarians' work.

DISCUSSION

Kingsborough library's adoption of Vue has provided an example of how an academic library has applied modular approaches to library programming projects. There are numerous pragmatic reasons why adopting a modular approach to code has been a good decision for the library. This section will briefly enumerate some advantages, in order to give an explanation as to why modularism has had a net benefit on the Kingsborough library. While these advantages are discussed here in an abstract way, they also often represent tangible, practical ways that modularized code has improved the Kingsborough library's code.

Testing

Modularized code offers a robust model for maintainability. Indeed, as Lavy and Rami note, "one of the key indicators for testing code quality is the level of modularity."[49] One of Lavy and Rami's insights is that modularity supports code testing, which is a key pillar of maintainability for most software projects. While the authors of smaller projects can often get away with not systematically writing tests for their code, it is essential for larger projects. While the (smallish) Kingsborough library website project did not use formal testing, the librarians certainly recognize its value and importance, as well as its strong ties to modularism. Modularism supports best practices in development by encouraging testable code.

Affordability

An important benefit of modularized workflows, pointed out by Schilling, is that modular approaches yield lower costs.[50] This is obviously important to any organization, but is particularly relevant in academic libraries, which typically have limited budgets and personnel resources. Library departments must do their best with the resources available. Modular programming helps by encouraging reusability, appropriate scoping, and small, maintainable components. In this way, it reduces development time. Modularism makes codebases more manageable with limited resources.

Maintainability

With a modular approach, a programmer only needs to understand one piece of code at a time in order to make changes to the codebase. There is immense value to this: the cognitive load of understanding a module will be much less than the cognitive load of understanding the entire project. Following Lin, and Gonçalves et al, modular projects can be contrasted with "monolith" projects.[51] The takeaway here is that modularism makes code much more accessible to more individual coders. It opens up the opportunity to maintain and work on the code to more people: a prospective maintainer needs to only understand a specific module—not the entirety of the project—to contribute constructively. This has a strong net positive effect on a project.[52] Indeed, Lavy and Rami go further, to describe the benefits of modular maintainability for novice programmers.[53] In a modularized system, librarians can maintain the parts of the code that they can make sense of. Parts that are not relevant can often be mostly ignored (perhaps for the time being) because the modular structure of the project permits the librarian to work only on the modules that are relevant to them.

Staffing

A related concern is that small libraries often do not have the luxury of staffing redundancy in key functional areas. At the Kingsborough library, each library function is managed by an individual librarian. Indeed, many of the librarians at the college have more than one functional area that they are responsible for. Practically speaking, this results in siloing.

Also, perhaps atypically, librarians at the Kingsborough library sometimes switch functional roles, resulting in a need to transmit knowledge between librarians—or from one silo to another—even more frequently. In this context, transmitting institutional knowledge is especially difficult, due to the lack of bandwidth (and budget) to cross-train librarians in different areas. As a result, there needs to be a way to transmit knowledge during staffing changes.

Modular code provides a way to address the challenges of functional silos. While the technical affordances of modularism for dealing with siloing are basically the same as those that promote maintainability (as described in the previous section), the institutional impetus in this case is different. Sharing responsibility for maintenance of technical systems is an important part of library resilience and responsiveness to staffing changes. Modularism plays a key role in increasing the ability of librarians to understand and maintain their library codebases as a team, rather than having it be solely the domain of one or two specialist librarians or programmers.

Librarians' Skill Sets

The last reason suggested here for adopting modular approaches in academic libraries has to do with the typical skill sets of librarians. Many librarians do not know programming languages, at least in much depth. This is not meant as a criticism, but rather—to be pragmatic about the situation—as a starting point for discussing the feasibility of maintaining library codebases.

Modularism helps with this by making maintenance accessible to many more librarians. Yet it is not a panacea for improving librarians' coding skills. Nor does it obviate the need for more complex pieces of the library infrastructure; those are—without question—going to require some more advanced programming knowledge at some point. But it does introduce an entry point for a curious librarian to begin working with a library codebase. It can open a door to getting started with programming, or to go further on one's programming journey. This can be an important step in developing librarians' technical skills.

CONCLUSION

Modularism is an important tool for building library code infrastructure. It can be used as a powerful method to push back against complexity. Librarians are often just trying our best to avoid the spaghetti code described by Davis, because, as Westra says, ultimately "spaghetti coding is not an option."[54] For a programmer today—or indeed at any time since the 1970s—writing modular code is an essential technique for producing good code. The benefits of modular code for maintainability have long been agreed upon by most programmers. Modularism is seen as encouraging simplicity and composability in code.

This paper has suggested, starting with a brief theoretical overview, how modularism can be seen as a type of Weberian modernism. It has attempted to show how library software is an heir to this rationalism, even today, over a century after Weber's death. While from a pragmatic perspective, modularism is a boon to librarians writing code, it is also necessary to be cognizant that it is closely related to many discontents with Weberian capitalism.

That Weber's "spirit of capitalism" would so directly affect academic libraries is not surprising or novel. While academic libraries are perhaps in general less "red in tooth and claw" about capitalism than many other organizations, the logics of rationalization are nonetheless profoundly woven into contemporary library software development in ways that are often unavoidable.[55] This paper only hopes to shed some light on one of the interesting ways in which this influence is propagated.

This paper has suggested some pragmatic best practices that arise out of a commitment to modularism in code and discussed some of the direct benefits that modular programming has for academic libraries. The hope is that the conceptual background provided here provides some grounding and context for those practices, and that it encourages librarians to think conscientiously about their work, and to consider adopting modular methods for themselves, their code, and their libraries, when appropriate.

ENDNOTES

[1] Melissa A. Schilling, *Strategic Management of Technological Innovation* (McGraw-Hill Education, 2017).

[2] Carliss Y. Baldwin and Kim B. Clark, "Managing in an Age of Modularity," *Harvard Business Review* 75, no. 5 (1997): 84–93.

[3] Herbert A. Simon, "The Architecture of Complexity," *Proceedings of the American Philosophical Society* 106, no. 6 (1962): 467–82; Frank F. Pieper, *Modular Management and Human Leadership* (Methods Press, 1958).

[4] Pieper, *Modular Management*, 12.

[5] Angela D. Buscalioni et al., "Modularity at the Boundary Between Art and Science," in *Modularity: Understanding the Development and Evolution of Natural Complex Systems*, ed. Werner Callebaut and Diego Rasskin-Gutman (MIT Press, 2009); Paul Duchscherer and Douglas Keister, *The Bungalow: America's Arts and Crafts Home* (Penguin Studio, 1995).

[6] David L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM* 15, no. 12 (1972): 1053–58, https://doi.org/10.1145/361598.361623; Jeff Maynard, *Modular Programming* (Petrocelli Books, 1972); Pieper, *Modular Management*; Arthur B. VanGundy, *Managing Group Creativity: A Modular Approach to Problem Solving* (American Management Associations, 1984).

[7] Lyudmila Kozhukhovskaya, Vladimir Baskov, and Anton Ignatov, "Modular Management of Indicators of Efficiency and Safety of Transportation Processes," *Transportation Research Procedia* 20 (2017): 361–66, https://doi.org/10.1016/j.trpro.2017.01.048.

[8] James L. Nevins, Daniel E. Whitney, and Thomas L. De Fazio, *Concurrent Design of Products and Processes: A Strategy for the Next Generation in Manufacturing* (McGraw-Hill, 1989); Schilling, *Strategic Management*.

[9] Dutch Mandel, "Ford Kills Modular Truck Plan for '90s," *Automotive News* (1989): 1; J. Paralikas et al., "Product Modularity and Assembly Systems: An Automotive Case Study," *CIRP Annals* 60, no. 1 (2011): 165–68, https://doi.org/10.1016/j.cirp.2011.03.009; Valerio Acanfora et al., "A Feasibility Study on Innovative Reinforced Modular Frames for Automotive Applications," *Macromolecular Symposia* 404, no. 1 (2022): 2100455, https://doi.org/10.1002/masy.202100455.

[10] Claus Rueffler, Joachim Hermisson, and Günter P. Wagner, "Evolution of Functional Specialization and Division of Labor," *Proceedings of the National Academy of Sciences* 109, no. 6 (2012): E326-E355, https://doi.org/10.1073/pnas.1110521109; Werner Callebaut and Diego Rasskin-Gutman, eds. *Modularity: Understanding the Development and Evolution of Natural Complex Systems* (MIT Press, 2009).

[11] Gregroy Currie and Kim Sterelny, "How to Think About the Modularity of Mind-Reading," *The Philosophical Quarterly* 50, no. 199 (2000): 145–60, https://doi.org/10.1111/1467-9213.00176.

[12] Max Weber, *The Protestant Ethic and the Spirit of Capitalism* (Charles Scribner's Sons, 1958).

[13] Bruno Latour, *We Have Never Been Modern* (Harvard University Press, 1993); Anna Lowenhaupt Tsing, *Friction: An Ethnography of Global Connection* (Princeton University Press, 2005); Baldwin and Clark, "Managing in an Age."

[14] Mark J. Greeven et al., "Why Companies Must Embrace Microservices and Modular Thinking," *MIT Sloan Management Review* 63, no. 1 (2021): 1–6, https://sloanreview.mit.edu/article/why-companies-must-embrace-microservices-and-modular-thinking/.

[15] Greeven et al., "Why Companies Must Embrace"; Baldwin and Clark, "Managing in an Age."

[16] Parnas, "On the Criteria."

[17] Daniel Davis, Jane Burry, and Mark Burry, "Understanding Visual Scripts: Improving Collaboration through Modular Programming," *International Journal of Architectural Computing* 9, no. 4 (2011): 364, https://doi.org/10.1260/1478-0771.9.4.361.

[18] Parnas, "On the Criteria"; Maynard, *Modular Programming*; Baldwin and Clark, "Managing in an Age"; David L. Parnas, "The Secret History of Information Hiding," in *Software Pioneers: Contributions to Software Engineering*, ed. Manfred Broy and Ernst Denert (Springer, 2002). For a discussion of object-oriented programming, see https://press.rebus.community/programmingfundamentals/chapter/objects-and-classes/.

[19] Andy Hunt and Dave Thomas, *The Pragmatic Programmer: From Journeyman to Master* (Addison-Wesley Professional, 1999).

[20] Maynard, *Modular Programming*, 6.

[21] Ilana Lavy and Rashkovits Rami, "The Circumstances in which Modular Programming Becomes the Favor Choice by Novice Programmers," *International Journal of Modern Education and Computer Science* 10, no. 7 (2018): 1, https://doi.org/10.5815/ijmecs.2018.07.01.

[22] Lavy and Rami, "The Circumstances."

[23] Lev Manovich, *Software Takes Command* (Bloomsbury, 2013).

[24] "apt," Debian, 2025, https://salsa.debian.org/apt-team/apt; "npm/cli," npm, GitHub, 2025, https://github.com/npm/cli; "pypa/pip," Python Packaging Authority, GitHub, 2025, https://github.com/pypa/pip; "Rubygems/Rubygems," RubyGems, GitHub, 2025, https://github.com/rubygems/rubygems.

[25] James Lewis and Martin Fowler, "Microservices: A Definition of this New Architectural Term," *MartinFowler.com*, March 25, 2014, https://martinfowler.com/articles/microservices.html; Sam Newman, *Building Microservices: Designing Fine-Grained Systems* (O'Reilly Media, 2015); Sam Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith* (O'Reilly Media, 2020).

[26] Newman, *Monolith to Microservices*.

[27] Victor Velepucha and Pamela Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," *IEEE Access* 11 (2023): 88339–58, https://doi.org/10.1109/ACCESS.2023.3305687.

[28] Anusha Reddy Guntakandla, "Microservices and Modular Architecture: Revolutionizing E-Commerce Scalability," *Journal of Computer Science and Technology Studies* 7, no. 4 (May 10, 2025): 133–37, https://doi.org/10.32996/jcsts.2025.7.4.15.

[29] Yu Gan and Christina Delimitrou, "The Architectural Implications of Cloud Microservices," *IEEE Computer Architecture Letters* 17, no. 2 (July 1, 2018): 155–58, https://doi.org/10.1109/LCA.2018.2839189.

[30] Anelis Pereira-Vale et al., "Security in Microservice-Based Systems: A Multivocal Literature Review," *Computers & Security* 103 (2021): 102200, https://doi.org/10.1016/j.cose.2021.102200; Paul Osman, *Microservices Development Cookbook: Design and Build Independently Deployable, Modular Services* (Packt Publishing, 2018).

[31] Kehua Miao et al., "A Microservice-Based Big Data Analysis Platform for Online Educational Applications," *Scientific Programming* (June 3, 2020): 1–13, https://doi.org/10.1155/2020/6929750; Julius B. Mathews et al., "Industrial Applications of a Modular Software Architecture for Line-Less Assembly Systems Based on Interoperable Digital Twins," *Frontiers in Mechanical Engineering* 9 (February 28, 2023): 1113933, https://doi.org/10.3389/fmech.2023.1113933; Lamine Lagsaiar et al., "Modular Software Architecture for Local Smart Building Servers," *Sensors* 21, no. 17 (August 29, 2021): 5810, https://doi.org/10.3390/s21175810; Shihabur Rahman Chowdhury et al., "Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges," *IEEE Network* 33, no. 3 (2019): 168–76, https://ieeexplore.ieee.org/document/8688711/; Björn Butzin, Frank Golatowski, and Dirk Timmermann, "Microservices Approach for the Internet of Things," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA): September 6–9, 2016 Berlin, Germany* (IEEE, 2016), https://doi.org/10.1109/ETFA.2016.7733707; Petar Krivic et al., "Microservices as Agents in IoT Systems," in *Agent and Multi-Agent Systems: Technology and Applications*, ed. Gordan Jezic et al., Smart Innovation, Systems and Technologies 74 (Springer Cham, 2018), 22–31, https://doi.org/10.1007/978-3-319-59394-4_3; Michel Krämer and Julia Senner, "A Modular Software Architecture for Processing of Big Geospatial Data in the Cloud," *Computers & Graphics* 49 (2015): 69–81, https://doi.org/10.1016/j.cag.2015.02.005.

[32] For a discussion of objects and classes, see https://press.rebus.community/programmingfundamentals/chapter/objects-and-classes/.

[33] Ernst Pisch et al., "M-Score: An Empirically Derived Software Modularity Metric," in *Proceedings of the 18th ACM/IEEE Symposium on Empirical Software Engineering and Measurement* (ACM, 2024), 382–92, https://doi.org/10.1145/3674805.3686697.

[34] Sasan Seydnejad, *Modular Programming with JavaScript: Modularize Your JavaScript Code for Better Readability, Greater Maintainability, and Enhanced Testability* (Packt Publishing, 2016).

[35] Closures are more of an advanced topic, but a decent explanation can be found at https://medium.com/@prashantramnyc/javascript-closures-simplified-d0d23fa06ba4.

[36] Davis et al., "Understanding Visual Scripts," 363.

[37] Weber, "Protestant Ethic," 135.

[38] Evan You, "The Progressive Framework," 2023, https://vuejs.org/.

[39] In this context, "progressive" basically means scalable, both up and down.

[40] For a discussion of encapsulation, see https://press.rebus.community/programmingfundamentals/chapter/encapsulation/.

[41] Seydnejad, *Modular Programming with JavaScript*.

[42] Jonathan Bradley, "Creating a Custom Queueing System for a Makerspace Using Web Technologies," *Code4Lib Journal* 55 (2023), https://journal.code4lib.org/articles/16876.

[43] Bradley, "Creating a Custom."

[44] Jim Hahn and Nathaniel Ryckman, "Modular Mobile Application Design," *Code4Lib Journal* 18 (2012), https://journal.code4lib.org/articles/7336.

[45] *Library Management System: Alma—Ex Libris*, Ex Libris, 2025, https://exlibrisgroup.com/products/alma-library-services-platform/.

[46] *LibGuides*, Springshare, 2025, https://www.springshare.com/libguides.

[47] Schilling, *Strategic Management*, 223.

[48] Seydnejad, *Modular Programming with JavaScript*, 4.

[49] Lavy and Rami, "The Circumstances," 1.

[50] Schilling, *Strategic Management*.

[51] Robert Lin, "Monolithic vs Modular," *Medium* (blog), November 3, 2016, https://medium.com/@berto168/monolithic-vs-modular-9b6d69684a2c; Nuno Gonçalves et al., "Monolith Modularization Towards Microservices: Refactoring and Performance Trade-Offs," in *2021 IEEE 18th International Conference on Software Architecture Companion* (IEEE, 2021), 1–8, https://doi.org/10.1109/ICSA-C52384.2021.00015.

[52] Mark Eaton, "On the Ethics of Working with Library Technology: The Case of the Open Journal Matcher," *Journal of Web Librarianship* 16, no. 1 (2022): 68–78, https://doi.org/10.1080/19322909.2021.2013387.

[53] Lavy and Rami, "The Circumstances."

[54] Davis et al., "Understanding Visual Scripts"; Erik Westra, *Modular Programming with Python: Introducing Modular Techniques for Building Sophisticated Programs Using Python* (Packt Publishing, 2016), 1.

[55] Alfred Tennyson, *In Memoriam* (University of Michigan Library, 2005), 126, http://name.umdl.umich.edu/ADH9165.0001.001.