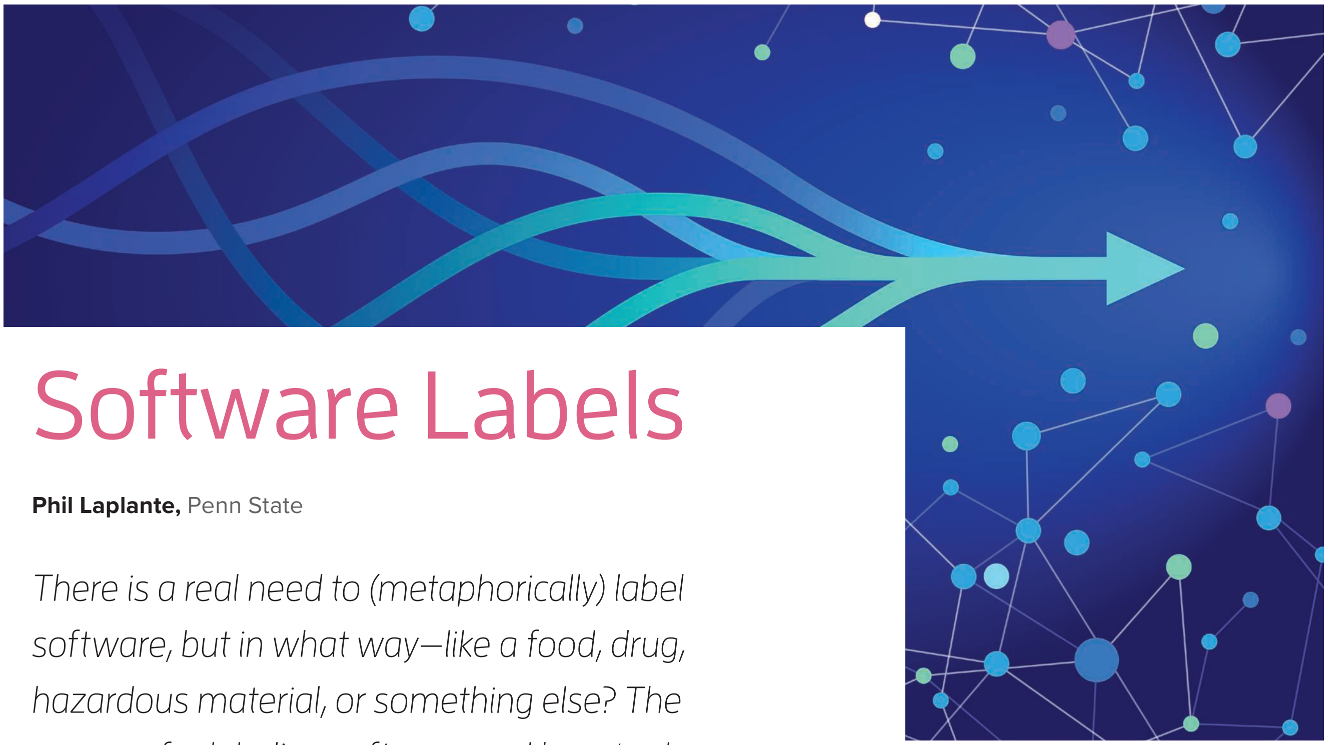


Software Labels

Phil Laplante, Penn State

There is a real need to (metaphorically) label software, but in what way—like a food, drug, hazardous material, or something else? The reasons for labeling software and how to do this are discussed.

If you are careful, you read the ingredients label on food packaging and menus to help ensure that you have a balanced diet and avoid risks, such as too much fat, sugar, cholesterol, and allergens. Likewise, when taking medications, you follow your doctor's and pharmacist's instructions and obey the warnings on the drug label. Finally, if you are self-aware, you constantly make safety-based choices, for example, when traveling to avoid a dangerous intersection or wearing appropriate gear when working in potentially dangerous conditions.

Wouldn't it be important, then, for systems designers to take similar precautions when using any software components and libraries, either off the shelf, custom built, free, or open source, especially in mission-critical systems? However, achieving this level of practice will not be easy. Open source software sites are often murky about the origins of the software contained therein (and who contributed), and custom software builders don't typically disclose the inner workings of their software unless contractually required.

Digital Object Identifier 10.1109/MC.2021.3102360
Date of current version: 22 October 2021

Labeling software components as to their origin, attributes, and "ingredients" is vitally important to help software and systems integrators meet their quality goals and reassure users of that software. In the wake of several ransomware attacks on critical infrastructure, the U.S. president issued an executive order with a number of mandates, including ways to secure the software supply chain and investigate ways to label software.[1] Let's discuss why it's important to label software and some of the ways this can be done.

HANDLE WITH CARE

Software in a deployed system can come from many sources—whether written from scratch in house or by an outside vendor, as a modified version of previously used software, autogenerated by tools, taken from open source, and so forth. Clearly, using software that has proven to be robust in prior deployment can help reduce the overall development cost. However, improper reuse can cause catastrophic failure. Knowing the properties of the software and how it should and should not be used is essential—just like checking for allergens in foods, observing the contraindications in drugs, and avoiding mixing dangerous compounds in chemicals. Improper reuse of software

EDITOR PHIL LAPLANTE
Penn State; plaplante@psu.edu

has led to serious catastrophes, such as the Therac-25 deaths and Ariane 5 rocket crash.[2]

Therefore, users of software components need to know important information about that software, and that's where labels come in. I see at least three (not mutually exclusive) ways to label software: as a food, drug, or hazardous material. Let's consider each of these possible approaches.

SOFTWARE LABELED LIKE FOOD

At a minimum, all publicly available software should be labeled like food. Just as any food item can be harmful (for example, via an allergic reaction), any software, when used in the wrong way, has the potential to cause harm. The U.S. Food and Drug Administration (FDA) requires that food be marked with a "Nutrition Facts" label and safe handling instructions to make food storage and use safer (Figure 1).

Other countries use similar labeling. The food label is intended to "make it easier [to make] informed food choices that contribute to lifelong healthy eating habits."[13] Likewise, the FDA requires fresh meat, poultry, and eggs to be labeled with safe handling instructions to provide important information on storage and preparation. This promotes the health, safety, and welfare of those who consume the food. Why not label software similarly to "make it easier for engineers to make informed software component choices that contribute to safe, reliable, and maintainable software"?

There are lots of candidates for a software ingredient label. These include the

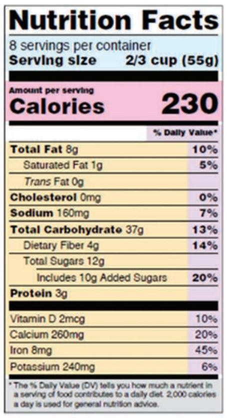
- amount of reused (modified and unmodified) code
- amount of new code (handwritten and autogenerated by tools)
- amount of open source code (and type of license)
- test and build scaffolding code included
- complexity (for example, measured by cyclomatic complexity, weighted cyclomatic complexity, Halstead's metrics)
- percentage and kinds of testing conducted as well as test coverage metrics
- number of authors or contributors to the component (ideally with their identities
- names and percentages of different programming languages used
- security testing done on the code
- as well as any other properties that contribute to reliability, safety, and security.

Similarly, the "safe handling instructions" label for the software could assert a secure chain of custody, testing, and storage recommendations or any other special conditions for its use. Of course, a "label" is a metaphor and really represents a database entry somewhere, but it's helpful to actually visualize these labels (Figure 2). Obviously, research, discussion, experience, and consensus are needed to determine the parameters for these labels.

SOFTWARE AS A DRUG

Every pharmaceutical comes labeled with a list of possible harmful side effects, contraindications, and other warnings for use by the pharmacist. Doctors also must know these facts through education and reference guides. Over-the-counter packaging includes this information in a more understandable form for the patient. Failure to follow any of these admonitions upon prescription, fulfillment, or administration could be disastrous.

Like a drug, certain software components can also potentially harmful when designed, built, or used incorrectly or not as intended. For example, certain software components might be suitable for use in office productivity, entertainment, or game software but not in insulin pumps or any automobiles. In these cases, the software could have a "software warning label" that includes warnings about certain operational profiles, notification of major feature changes, and recommendations on appropriate uses.



Nutrition Facts

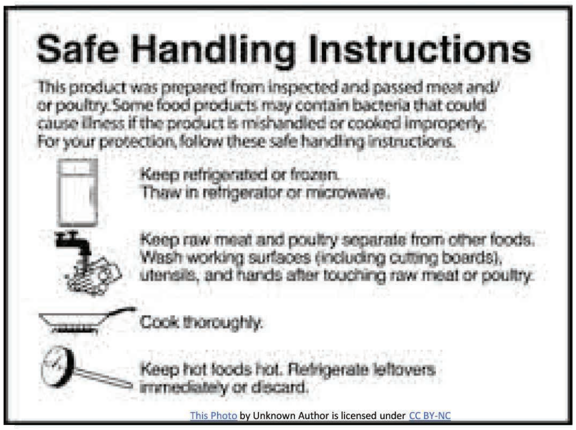
8 servings per container
Serving size **2/3 cup (55g)**

Amount per serving
Calories **230**

	% Daily Value*
Total Fat 8g	10%
Saturated Fat 1g	5%
Trans Fat 0g	
Cholesterol 0mg	0%
Sodium 160mg	7%
Total Carbohydrate 37g	13%
Dietary Fiber 4g	14%
Total Sugars 12g	
Includes 10g Added Sugars	20%
Protein 3g	
Vitamin D 2mcg	10%
Calcium 260mg	20%
Iron 8mg	45%
Potassium 240mg	6%

* The % Daily Value (DV) tells you how much a nutrient in a serving of food contributes to a daily diet. 2,000 calories a day is used for general nutrition advice.

(a)



Safe Handling Instructions

This product was prepared from inspected and passed meat and/or poultry. Some food products may contain bacteria that could cause illness if the product is mishandled or cooked improperly. For your protection, follow these safe handling instructions.

Keep refrigerated or frozen.
Thaw in refrigerator or microwave.

Keep raw meat and poultry separate from other foods. Wash working surfaces (including cutting boards), utensils, and hands after touching raw meat or poultry.

Cook thoroughly.

Keep hot foods hot. Refrigerate leftovers immediately or discard.

This Photo by Unknown Author is licensed under CC BY-NC

(b)

FIGURE 1. (a) A nutrition facts label and (b) food safe handling instructions. These images are for education purposes only. These labels do note meet the labeling requirements described in the Code of Federal Regulations (CFR) (21 CFR 101.9). (Source: U.S. Department of Agriculture.[3])

For fun, I created a prototype software warning label by marking up a drug warning label (Figure 3). Here, I replaced the notifications for "Dosage and Administration" with "Recommended Scenarios." The sections for "Adverse Reactions" and "Drug Interactions" on the FDA label were changed to "Incompatibility" and "Side Effects," respectively. The heading "Contraindications" was replaced with "Excluded Scenarios." Software and systems engineers would have important information at hand if every component came with such a software warning label.

SOFTWARE AS A HAZARDOUS MATERIAL

The U.S. Cybersecurity Infrastructure and Security Agency identifies 16 critical infrastructure sectors: chemical, commercial facilities, communications, critical manufacturing, dams, defense industrial base, emergency services, energy, financial services, food and agriculture, government facilities, health care and public health, information technology, nuclear reactors, transportation systems, and water and wastewater systems.[5] These sectors envelop the class of "critical systems."

"Critical software" is any software embedded in a critical system or system that interacts or has the potential to interact (even indirectly) with a critical system. When software is used in major critical systems, failure, whether accidental or through hacking, can cause significant harm to the public or infrastructure. Therefore, instead of labeling critical software like a food or drug, we should label it like a hazardous chemical.

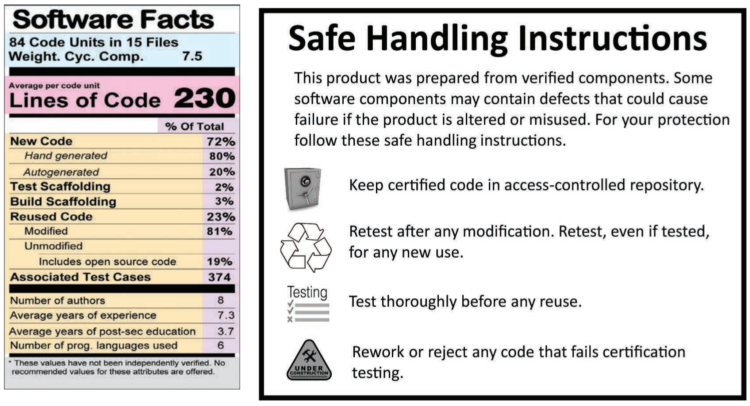
The U.S. Occupational Safety and Health Administration (OSHA) Hazard Communication Standard requires that the chemical manufacturer, distributor, or importer provide safety data sheets (SDSs) for each hazardous chemical to downstream users to communicate information on these hazards. The SDS includes "information, such as the properties of each chemical; the physical, health, and environmental health hazards; protective measures; and safety precautions for handling, storing, and transporting the chemical." It also includes "physical and chemical properties, stability and reactivity information, toxicological information, exposure control information, and other information including the date of preparation or last revision."

The SDS tells what to do in the case of a disaster, for example, if there is a fire (does the chemical produce toxic fumes?) or the substance is ingested. There isn't enough room in this article to share a sample SDS, but the specifications for one can be found at the OSHA website.[6]

Chemical SDSs are quite detailed and complex, and understanding them requires expertise that the average person does not have. The SDS database also needs to be religiously maintained and updated in a way that makes the documents accessible in case of an emergency.

Continuous training is usually associated with the regular changes in an SDS. Usually, in large laboratories, one or more persons are identified as a "Safety Officer," who is thoroughly and continuously trained and charged with dealing with the SDSs for all of the substances in the lab as well as preparing to respond to a disaster. This description sounds curiously like the way that critical software needs to be handled.

Producers of critical software should be required to create a kind of SDS with analogous warnings of potential interactions with other software, possible side effects of execution, security vulnerabilities, limitations, exclusions, known hazards and so on. Critical software should not be in the hands of the untrained. Software engineers working on critical software need to be specially and continuously trained to maintain the database of software SDSs.



Software Facts

84 Code Units in 15 Files
Weight. Cyc. Comp. 7.5

Average per code unit
Lines of Code 230

	% Of Total
New Code	72%
Hand generated	80%
Autogenerated	20%
Test Scaffolding	2%
Build Scaffolding	3%
Reused Code	23%
Modified	81%
Unmodified	
Includes open source code	19%
Associated Test Cases	374
Number of authors	8
Average years of experience	7.3
Average years of post-sec education	3.7
Number of prog. languages used	6

* These values have not been independently verified. No recommended values for these attributes are offered.

Safe Handling Instructions

This product was prepared from verified components. Some software components may contain defects that could cause failure if the product is altered or misused. For your protection follow these safe handling instructions.

Keep certified code in access-controlled repository.

Retest after any modification. Retest, even if tested, for any new use.

Testing

Test thoroughly before any reuse.

UNDER CONSTRUCTION

Rework or reject any code that fails certification testing.

(a) (b)

FIGURE 2. An (a) ingredient label and (b) safe handling instructions for software. These images are for discussion purposes only. These indicators are all subject to debate and must be customized to the application/programming language(s). Cyc. Comp.: cyclomatic complexity; post-sec: post-secondary; prog.: programming.

SOFTWARE WARRANTIES

Yet another type of label is suggested by the one attached to the window of a new car for sale. That label gives details about where the car was manufactured, the type of engine, the fuel efficiency based on the type of driving (city or highway), the tire manufacturer and type, and the human comforts inside (such as leather seats, air conditioning, and so on), among other characteristics. These are all features or attributes of the product.

There will also be a statement on this label about the warranty—for example, five years or 50,000 mi—and it will explain what parts of the car that warranty applies to. It might only be the engine, or it could be bumper to bumper, meaning any part of the car. For used cars, a label might be still posted, but it might be a weaker warranty or none at all. It may also say that you need to pay an additional fee if you want a warranty or longer warranty.

In 2000, Voas[7] proposed the idea of a limited software warranty, which is essentially a certificate that describes "something" about the quality of a software artifact. This could be as simple as "This software is 50% reliable" or "This software was tested 100 times using this operation profile, and here are the results of that testing."

The point of proposing a limited software warranty was to move software products away from being feared as "Buyer beware" and "What you see is what you get." In essence, the industry should turn from the assumption that commercial software, not open source, must be viewed as a black box with no associated pedigree. In essence, this limited warranty would be a virtual window sticker on the software.

LABELED SOFTWARE AS A COMPETITIVE ADVANTAGE

The labeling of software does not give away intellectual property, just as a food label does not reveal any secret recipes. A label only discloses important information about what is in the food, and, in the case of the software label, it exposes important properties of the code that are conducive to safety, security reliability, and so on, which are especially important in critical systems. Giving these assurances does not divulge any trade secrets or intellectual property. In fact, it is a competitive advantage.

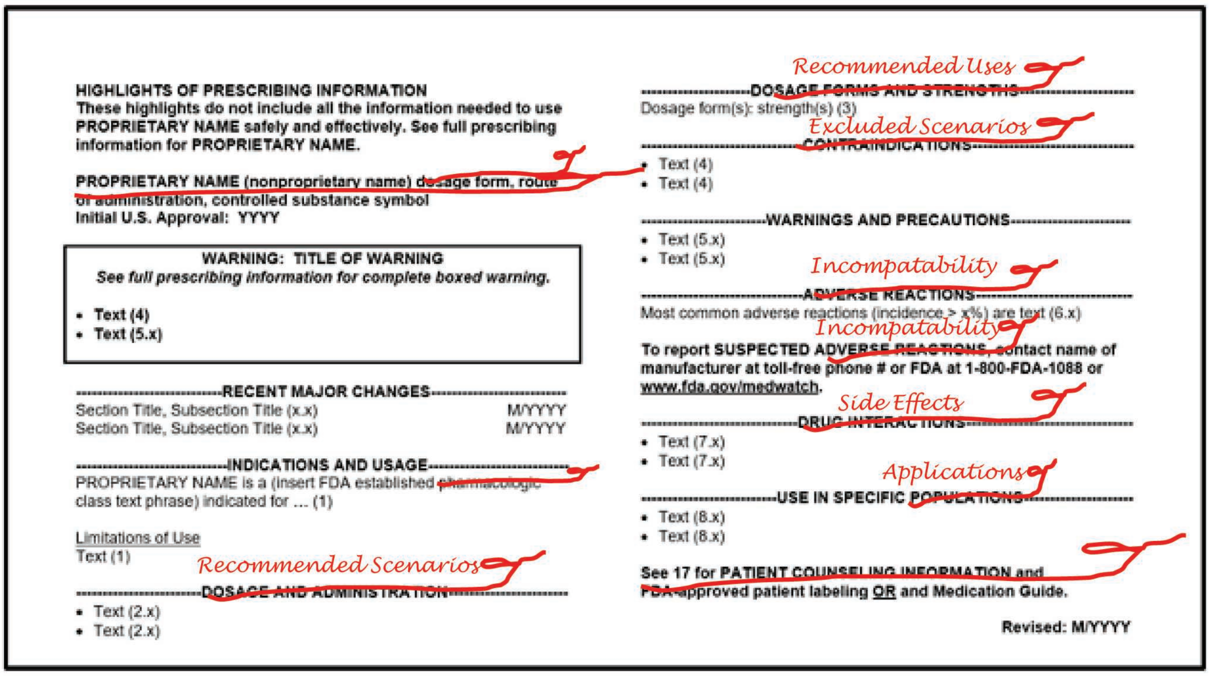
Each software provider can advertise its own "special blend" of testing, test coverage, complexity, new and reused, open access, and so on. Then, the software component consumer can decide based on these qualities that the system is safe, reliable, trustworthy, and so on.

Of course, these qualities will have different limits based on the application domain, and the relevant standards body or regulatory agency in the particular application domain can set these minimum requirements, maximum limits, and so on for these labeled quantities. Since these are not physical labels, a repository for the software products and their labels as well as "recommended daily allowances" can be kept and secured by an appropriate agency.

MORE THAN JUST LABELING

Just as we need to ensure certain qualities of the food ingredients, preparation facilities, and processes used as well as those who do the preparation, we need to do the same with respect to the safety, security, privacy, reliability, and so on for software. To ensure safety and public trust in software systems and components, especially for critical software, we need labeling and a whole lot more. Trust needs to be established in the product, process, and people involved, particularly with regard to security.

There are many efforts to meet these objectives. For example, the U.S. National Institute for Standards and Technology (NIST) developed a "Secure Software Development Framework,"[8] which includes several such recommendations, for example, "verify third-party software complies with security



HIGHLIGHTS OF PRESCRIBING INFORMATION
These highlights do not include all the information needed to use PROPRIETARY NAME safely and effectively. See full prescribing information for PROPRIETARY NAME.

~~PROPRIETARY NAME (nonproprietary name) dosage form, route of administration, controlled substance symbol~~
Initial U.S. Approval: YYYY

WARNING: TITLE OF WARNING
See full prescribing information for complete boxed warning.

- **Text (4)**
- **Text (5.x)**

-----RECENT MAJOR CHANGES-----
Section Title, Subsection Title (x.x) M/YYYY
Section Title, Subsection Title (x.x) M/YYYY

-----INDICATIONS AND USAGE-----
PROPRIETARY NAME is a (insert FDA established ~~pharmacologic~~ class text phrase) indicated for ... (1)

Limitations of Use
Text (1)

Recommended Scenarios
-----~~DOSAGE AND ADMINISTRATION~~-----
- Text (2.x)
- Text (2.x)

Recommended Uses
-----~~DOSAGE FORMS AND STRENGTHS~~-----
Dosage form(s): strength(s) (3)

Excluded Scenarios
-----~~CONTRAINDICATIONS~~-----
- Text (4)
- Text (4)

-----WARNINGS AND PRECAUTIONS-----
- Text (5.x)
- Text (5.x)

Incompatability
-----~~ADVERSE REACTIONS~~-----
Most common adverse reactions (incidence > x%) are text (6.x)

Incompatability
To report SUSPECTED ~~ADVERSE REACTIONS~~, contact name of manufacturer at toll-free phone # or FDA at 1-800-FDA-1088 or www.fda.gov/medwatch.

Side Effects
-----~~DRUG INTERACTIONS~~-----
- Text (7.x)
- Text (7.x)

Applications
-----USE IN SPECIFIC ~~POPULATIONS~~-----
- Text (8.x)
- Text (8.x)

~~See 17 for PATIENT COUNSELING INFORMATION and FDA~~-approved patient labeling OR and Medication Guide.

Revised: M/YYYY

FIGURE 3. A partially marked-up FDA drug warning label converted to a prototype software warning label. (Source: FDA.[4])

requirements." Also, the U.S. National Telecommunications and Information Administration introduced a framework to trace software lineage ("pedigree" or "provenance") called a *software bill of materials* (*SBOM*).[9] Loosely, an SBOM is a dependency diagram of the precursors and constituent parts of a piece of software. In other words, the software supply chain is, at least, partially identified.

NIST has also published NISTIR 8060, "Guidelines for the Creation of Interoperable Software Identification Tags," an XML-based tagging system for software components.[10] The Linux Foundation Core Infrastructure Initiative offers a Best Practices badge as "a way for Free/Libre and Open Source Software (FLOSS) projects to show that they follow best practices."[11] And the Underwriter's Laboratory (UL) Cybersecurity Assurance Program[12] is intended to be a combination of label and development guidelines. However, these efforts, while impressive and valuable, are independent, not always harmonized, and not widely adopted.

ASSIGNING RESPONSIBILITY

There needs to be a consistent and coordinated way to assess the level of risk in software and then decide if it needs to be labeled like a food, drug, or hazardous material or in more than one way. Appropriate limits and allowances must be established. Then, there needs to be a way to certify and securely store the software along with the metaphorical labels in a way that is trusted and available to the public, or at least trusted entities. All of the other efforts to establish a software pedigree, such as an SBOM, software tags, assurance frameworks, and more, need to be coordinated.

This begs the following questions: Which agency or agencies will have the responsibility for these activities? Should there be a "Software FDA"? Is it an international body? Who administers this body? Can it issue fines or close down software producers for violations? Who are the inspectors? What is the role of professional societies, government, domain discipline bodies, and so on?

Whether we treat software like a food, drug, hazardous material, car, or something else, much more study, discussion, and consensus are needed. Whatever the approach, constructing a universal framework for labeling software and an appropriate agency is an urgent matter. ◼

REFERENCES

1. "Executive order on improving the nation's cybersecurity," The White House, May 12, 2021. Accessed: June 5, 2021. [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improvilng-the-nations-cybersecurity/
2. E. Wong, X. Li, and P. Laplante, "Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures," *J. Syst. Softw.*, vol. 133, pp. 68–94, Oct. 2017. doi: 10.1016/j.jss.2017.06.069.
3. "Labeling," Food Safety and Inspection Service, U.S. Dept. of Agriculture, Washington, D.C. Accessed: June 5, 2021. [Online]. Available: https://www.fsis.usda.gov/sites/default/files/media_file/2021-03/FPLIC_2a_Labeling.pdf
4. "Prescription drug labeling resources," U.S. Food and Drug Administration, Washington, D.C. Accessed: June 5, 2021. [Online]. Available: https://www.fda.gov/drugs/laws-acts-and-rules/prescription-drug-labeling-resources
5. "Critical infrastructure sectors," Cybersecurity & Infrastructure Security Agency, Washington, D.C. Accessed: June 5, 2021. [Online]. Available: https://www.cisa.gov/critical-infrastructure-sectors
6. "Hazard communication standard: Safety data sheets," Occupational Safety and Health Administration, Washington, D.C. Accessed: June 5, 2021. [Online]. Available: https://www.osha.gov/Publications/OSHA3514.pdf
7. J. Voas, "Limited software warranties," in *Proc. IEEE Conf. Workshop on Eng. Comput.-Based Syst.*, Apr. 2000, pp. 56–61.
8. "Secure software development framework," Computer Security Resource Center, NIST, Gathersburg, MD. Accessed: June 5, 2021. [Online]. Available: https://csrc.nist.gov/Projects/ssdf
9. "Software bill of materials," National Telecommunications and Information Administration, U.S. Dept. of Commerce, Washington, D.C. Accessed: June 5, 2021. [Online]. Available: https://www.ntia.gov/SBOM
10. "Guidelines for the creation of interoperable software identification (SWID) tags," Computer Security Resource Center, NIST, Gathersburg, MD, NISTIR 8060, Apr. 2016. Accessed: June 5, 2021. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8060/final
11. "CII Best Practices badge program," CII Best Practices. Accessed: June 11, 2021. [Online]. Available: https://bestpractices.coreinfrastructure.org/en
12. "Cybersecurity assurance and compliance," UL. Accessed: June 11, 2021. [Online]. Available: https://www.ul.com/services/cybersecurity-assurance-and-compliance
13. "The new nutrition facts label: What's in it for you?" U.S. Food and Drug Administration, Washington, D.C. Accessed: Sept. 13, 2021. [Online]. Available: https://www.fda.gov/food/nutrition-education-resources-materials/new-nutrition-facts-label

PHIL LAPLANTE is professor of software and systems engineering at Penn State, Malvern, Pennsylvania, 19355, USA, a Fellow of the IEEE, and an associate editor in chief of Computer. Contact him at plaplante@psu.edu.