# CSN-254
# Software Engineering
# Critical Review - "Software Labels"

Group-4


| Name | Roll No | Contribution |
|---|---|---|
| Meet Sindhav | 22114053 | Discussion, articulation and final documentation |
| Mohammed Haaziq | 22114055 | Reviewing document and feedback |
| Aditya Mundada | 22114058 | Discussion and articulation |
| Nayan Kakade | 22114060 | Reviewing document and suggesting changes |
| Polasa Hariharan | 22114068 | Reviewing document and feedback |
| Roopam Taneja | 22125030 | Reviewing document and suggesting changes |


## SUMMARY


The article emphasizes the need for **labelling software components**, similar to food or drug labelling, to ensure transparency and safety in software usage, especially in critical systems. It discusses the challenges in the software supply chain, such as murky origins and lack of disclosure, and suggests labelling as a solution. Three approaches to labelling software are proposed: labelling software like **food, drug, or hazardous material**. The article explores the potential elements of each label, such as reused code, testing metrics, and security measures. It also discusses the concept of software warranties and how they could provide assurances of software quality. Furthermore, the article suggests that labelled software could be a competitive advantage, as it would allow consumers to make informed decisions. It highlights various efforts and initiatives aimed at improving software quality, such as **Secure Software Development Frameworks** and **Software Bill of Materials**. The article concludes by addressing the need for a coordinated approach to software labelling and certification, including assigning responsibility to regulatory bodies and establishing standards for labelling software components.


## KEY POINTS

- The document “Software Labels” discusses the need to label software products and tell the customer important aspects of the software when the software is made available to use so that it can be used safely and reliably
- Labeling software components means mentioning their origin, attributes, and “ingredients” when making the software available to use. It is vitally important as it helps software and systems integrator meet their quality goals and reassure users of that software.
- 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.

- We categorize software products just like food products into three major categories: Food, drugs and hazardous chemical.
- **Labelling as Food:** At a minimum, all publicly available software should be labelled 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 food label includes two parts- software ingredients and safe handling instructions.

  Just like the ingredients label in food items, the software must also have a software ingredients label. Software ingredients label includes: the amount of reused code, amount of new code and its origin, amount of open source code and type of license, test and build scaffolding code included, complexity of code, percentage and kinds of testing conducted as well as test coverage metrics, number of authors or contributors to the component, names and percentages of different programming languages used, security testing done on the code and other non-functional metrics relating to reliability, safety, and security.

  Safety handling instructions include recommendations on how to securely get custody of a software and test it as well as special conditions when it must not be used.
- **Labelling as Drugs:** Like a drug, certain software components can also be potentially harmful when designed, built, or used incorrectly or not as intended.

  Software classified in this category must have a software warning label. This label tells us that the software is suitable to use in which type of operation. For e.g., a component suitable to be used in an office management app is not suitable in a game. Thus, it tells us where the software is "incompatible", and its "side effects". This label recommends us appropriate users and notifies of major feature changes.
- **Labelling as hazardous chemicals:** Software embedded in a critical system or any as system that interacts with a critical system are termed as "Critical Software". Failures in such systems can have catastrophic consequences. Hence, such software can be labeled as hazardous chemicals.

  Producers, distributors or importers of hazardous chemicals are required to provide safety data sheets (SDSs). 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. 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.

  SDSs, being quite detailed and complex, requires expertise that an average person does not have. Similarly, such documents created for software should be maintained and updated by specially and continuously trained software engineers.
- The labeling of software does not give away intellectual property, just as a food label does not reveal any secret recipes. It exposes important properties of the code that are conducive to safety, security reliability, and so on, which are especially important in critical systems. It can turn out to become a competitive advantage.
- In addition to labels, we need a whole lot more to ensure safety and public trust in software. There are many efforts to reach these objectives, some of which are "Secure Software Development Framework", software bill of materials (SBOM), "Guidelines for the Creation of Interoperable Software Identification Tags", and "Underwriter's Laboratory (UL) Cybersecurity Assurance Program".

## POSITIVE ASPECTS

This article stresses on the importance of software labels and the ways in which this can be implemented. Following points show some of the positive aspects of this article:

- **Enhanced Safety and Reliability:**

  Labeling software by mentioning its attributes and features as well as analysis of its performance metrics can help users understand the potential risks associated with using a particular software. This transparency creates a safe and reliable environment for software deployment. This extra safety is especially useful in critical pieces of software.

- **Cost Reduction Through Proper Reuse:**

  The categorization of software and the inclusion of labels can guide developers and integrators in choosing robust components that have a proven track record. This can lead to cost reduction by avoiding unnecessary development efforts and potential failures.

- **Developing User Trust:**

  Just as labelling food products builds user trust because the user knows what he is buying, the software labeling system also provides a level of assurance to users regarding the quality, safety, and security of the software they are using. This enhances trust and confidence in the software industry.

- **Competitive Advantage:**

  By providing detailed information about the properties of software without revealing intellectual property, companies can establish a competitive advantage as they will come out as transparent and user friendly. Users may thus prefer software with clear labeling, especially in critical systems where safety and reliability are paramount.

- **Encourages Best Practices:**

  The emphasis on labeling encourages developers to follow best practices in software development, including proper testing, more focus on efficiency, security measures, and documentation. This can lead to overall improvements in software quality.

- **Inducing Industry Standardization:**

  The practice of labeling all software products will be beneficial in standardizing all products in the software industry where further regulations may be developed to allow only products above and at par with benchmark qualities to get authorizations for usage.

## SHORTCOMINGS AND PROPOSED SOLUTIONS

While the concept of software labelling to categorize software components into different classes (like food, drugs, etc.) has its merits, there are potential loopholes and challenges that need to be considered. Here are some of the loopholes and proposed solutions:

- **Complexity and Expertise:**
  ***Problem:*** Creating and maintaining software labels, especially for critical systems, requires a high level of expertise and understanding of the system. This may pose a challenge as it demands specialized knowledge that not all software developers or engineers may possess.
  ***Solution:*** Proper training must be provided, and a dedicated team must be created that monitors software

labels. Tools that help in documenting such labels must be made as user friendly as possible. A way could be to create software that allows developers to simply fill in various aspects of the component like a form which generates proper XML documentation to be attached with the component automatically. This makes the implementation of the concept convenient.

- **Prone to Vulnerabilities:**
  ***Problem:*** While labeling is intended to provide transparency, there is a risk that some parties may misuse the information for malicious purposes. Hackers or adversaries might exploit the disclosed information to identify vulnerabilities or weaknesses in the software.
  ***Solution:*** A tiring system must be maintained within labels where certain sensitive labels can only be accessed by authorized entities and not all end users. This makes it very secure and prevents leaking of important intellectual property to hackers.

- **Burden on Cost:**
  ***Problem:*** Requiring detailed labels, similar to safety data sheets for hazardous chemicals, could introduce bureaucratic burdens for software producers. This may slow down the development process and increase the overall cost of software production.
  ***Solution:*** Only critical software must be made to undergo a rigorous labelling process. Other software products can be labelled only on basic parameters that assure quality, but extensive documentation is not required for them.

- **Ever Evolving Nature of Software:**
  ***Problem:*** Software is dynamic and constantly evolving, on other hand, food products and drugs rarely change once they are manufactured. Keeping labels updated with the latest information might be challenging, and outdated labels may provide a false sense of security to users.
  ***Solution:*** An automated process can be implemented which provides timely updates when changes occur. It should also track labels, versions and patches of software used.

- **Impractical for Non-Critical Software:**
  ***Problem:*** The proposed labeling system may be more applicable and beneficial for critical systems; however, it is not salable to all software. For non-critical software, the effort and resources required to implement such labeling might outweigh the perceived benefits.
  ***Solution:*** Different labelling can be done based on the intended use and impact of the software. More detailed labelling efforts should be followed for critical systems while keeping requirements lighter for non-critical software. Such standards should be formulated by concerned regulated body and adopted by the industry.

- **Requirement of a Regulatory Body:**
  ***Problem:*** To implement and enforce this idea of software labels, a central regulatory body is essential which should have the authority to examine and test whether the software meets the required quality parameters, and the labelling follows specified rules and norms. Companies may not allow such examinations as it may lead to the leaking of trade secrets. Also setting up such a regulatory body will be tough and require a lot of resources and spending by Central governments and public organizations worldwide.
  ***Solution:*** Encouraging regulatory bodies, industry associations, and market leaders to advocate for and adopt labelling standards may help in its implementation. Also, incentives can also be provided for compliance and stakeholders should be educated about the benefits of labelling.

In conclusion, while the labeling of software brings forth positive aspects such as enhanced safety and user reassurance, there are challenges related to complexity, potential misuse, bureaucratic burden, the dynamic nature of software, and limited impact on non-critical software. Striking the right balance between transparency and practicality will be crucial for the successful implementation of this technique.

## TOOLS AND TECHNIQUES FOR IMPLEMENTING SOFTWARE LABELS

- **SWID tag:** A SWID (Software Identification) tag is a standardized **XML format** for a set of data elements that identify and describe a software product. It was developed by the **International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC)**. SWID tags assign a unique identifier to each software component. This identifier helps in distinguishing one software component from another. This unique identification is essential for accurate labelling and tracking of software components. These tags also include various software version details and licensing information as well as dependencies. This helps in identifying software contributors and software component types which help us in judging the security features and risks associated with a component. Each tag has the following parts:

  1. The ***Software Identity*** element is the root element representing the software entity.
  2. The ***Meta*** element contains metadata information such as activation status, last modified timestamp, publisher, and suppression status.
  3. The ***Link*** element provides a link to the schema definition for the SWID tag.
  4. The ***Evidence*** element includes information about the installation, such as the type of evidence (install) and the location.
  5. The ***Software Licences*** element summarizes licensing information associated with the software.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SoftwareIdentity
    xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
    name="ExampleSoftware"
    version="1.0.0"
    tagId="example-tag-123"
    uniqueId="urn:uuid:550e8400-e29b-41d4-a716-446655440000">

    <!-- Metadata Section -->
    <Meta
        xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
        activationStatus="active"
        lastModified="2024-03-04T12:30:00Z"
        publisher="ExamplePublisher"
        suppress="false"
    />

    <!-- Link Section -->
    <Link
        rel="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
        href="http://example.com/example-software.xsd"
    />

    <!-- Evidence Section -->
    <Evidence
        xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
        id="evidence-1"
        type="install"
        location="C:\Program Files\ExampleSoftware\"
    />

    <!-- Software Licenses Section -->
    <SoftwareLicenses
        xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
        summary="Licensed under Example License"
    />
</SoftwareIdentity>
```

**Figure 1:** SWID tag

- **SBOM (Software Bill of Materials):** 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. It lists all the **open source and third-party components** present in the software; their versions used in codebase; and their patch status. Idea of SBOM comes from manufacturing, where a Bill of Materials is an inventory detailing all items included in product. Similarly, Bill of Materials that includes an inventory of third-party and open-source components to ensure code is high-quality, compliant, and secure. A powerful **Software Composition Analysis (SCA) tool** such as **Black Duck®** can generate a complete open source SBOM, in addition, a SCA can provide this information on continuous basis, ensuring that one has the most up-to-date picture of open-source risks.

## REFERENCES


- https://www.synopsys.com/blogs/software-security/software-bill-of- materials-bom.html
- https://ieeexplore.ieee.org/abstract/document/1007518/similarsimilar
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=6835311
- https://ieeexplore.ieee.org/document/9585141
- https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf