

Including Information System Control Issues in Responsible Digital Policies

François Pellegrini, francois@pellegrini.cc

Version v1.0a, 12/17/2025

1 Context

“Responsible digital” aims to reduce the social, environmental, and economic footprint of digital technology by acting on both technologies and uses. Since 75 to 80% of the environmental footprint of digital technology comes from the manufacturing stage of hardware, increasing the lifespan and reusability of hardware is a dominant factor in reducing the environmental footprint of digital technology. However, an active digital device (i.e., one with programmable control logic) cannot operate without the associated software that controls it.

1.1 Imposed Renewal

While technological progress may justify the early retirement of a functional fleet, in case of significant performance improvement in the next generation (one can think of replacing CRT monitors with LCD screens), a “responsible” renewal decision must remain in the hands of the people concerned, who have to bear the cost and externalities. This decision cannot be dictated by suppliers, based on interests sometimes contrary to those of the affected organizations. However, in the past, such dictats have occurred, as with the release of Windows Vista, whose resource voracity (both in computing power and memory) forced too many organizations to replace their computer equipment, even recent one, with hardware powerful enough to run this new software.

1.2 Sources of Obsolescence

There are numerous obstacles to the continued operation of digital systems.

The first is hardware obsolescence. It can take the form of choosing components with a short lifespan during the manufacture of equipment, or choosing architectures and manufacturing processes that hinder repairability (for example, plastic welding instead of screwing). This is now a well-identified issue among responsible digital stakeholders and public authorities, who seek to influence the market by producing repairability indicators and imposing minimum values for future purchases, for example.

The second is software obsolescence. The latter is currently insufficiently taken into account in “responsible digital” policies, if at all. However, its effects on hardware disposal are considerable.

1.2.1 Programmed Software Obsolescence

Software obsolescence is mostly “programmed obsolescence”: while, in some cases, the introduction of more complex algorithms or more data-intensive mathematical models requires immediately acquiring more powerful hardware, in the majority of cases, it is the end of software support by the publisher or integrator that signals the death of a system. This death can be slow, materialized by a progressive loss of compatibility with new versions of interdependent surrounding systems, which will cause increasing inconvenience in use; this is the case, for example, when the lack of support for new file format versions forces users to convert to older versions for the system to work. This death can also be sudden, when upgrading surrounding software causes an abrupt end to compatibility. This is also the case when a vulnerability is discovered in a version of software that is no longer maintained: system operators, faced with such an unpredictable and sudden occurrence, must then immediately disconnect the system in question. In order not to find themselves in this situation and to reduce the attack surface, systems are therefore mostly retired and replaced as soon as their software maintenance is no longer assured. The existence of a commercial long-term support (LTS) offer is not, however, sufficient to solve the problem: these offers are generally also limited in time, and increase the cost of systems compared to their replacement.

1.2.2 Triggered Software Obsolescence

Programmed obsolescence can also be “triggered obsolescence”. This refers to the fact that obsolescence occurs suddenly, by the decision of a third party, such as when servers feeding terminals are shut down: these are rendered irreversibly unusable. This is notably the case with services attached to game consoles or, recently, Apple's remote triggering of a mechanism rendering inoperable all devices sold in Russia whose replacement components were considered by this company as “non-authentic”. This example illustrates the use of component marking techniques using coded elements to individually trace their origin. This practice also makes it possible to prevent the marketing of compatible components, as the exact copy of these elements, necessary for their acceptance by the system, is prosecuted as a violation of the company's copyright on these formal elements (this misuse of copyright is called “para-copyright” in the literature).

Triggered obsolescence constitutes a major ecological and strategic risk for organizations, as it is unpredictable and constitutes a serious threat to information system control, thus carrying digital sovereignty issues.

1.3 Fighting Programmed Obsolescence

Programmed obsolescence is avoidable. It results from excessive control by software publishers over the features and roadmap of their products. This imbalance prevents customers and users from asserting their right to “undisturbed enjoyment” of their computer systems, strikingly demonstrating the relevance of Lawrence Lessig's aphorism, “Code is law”: what has been programmed by a third party is imposed on users as a law from which one cannot derogate.

Programmed obsolescence is a proven and costly phenomenon, but it is not measured in digital usage studies, particularly in the mobile terminal sector, although they are largely subject to it. The still currently very (too) rapid renewal of mobile terminals masks the phenomenon, with users

changing equipment well before publishers end software support. Only the rare users wishing to extend their equipment's life beyond five to seven years find themselves confronted with the end of support for their operating systems, which will lead to progressive difficulties in installing new versions of mobile applications.

However, this subject has already been identified, without being translated into action. Thus, the ADEME report on digital technology mentions that: “Extending the lifespan of computer equipment, encouraged by the use of less resource-intensive operating systems like Linux, could reduce electronic waste by 30% to 50%”, by gaining 2 to 4 additional life years on terminals. A study on the respective performance of operating systems indicates that, for equal tasks, a Linux system saves 15 to 20% energy compared to a Microsoft Windows system. The use of Linux is often rejected on the argument that these systems are minority in number and therefore would not be able to offer the range of application solutions found under Microsoft Windows, for example. This is partially true, but the situation is rapidly evolving. In addition to the existence of emulators allowing the execution under Linux of applications designed for Microsoft Windows, a strong signal comes from the rapid breakthrough of Linux in India, where it represented, mid-2024, 15% of the workstation market. The emergence of such a fleet, in a country of 1.7 billion inhabitants, is likely to push publishers to offer their software natively on this platform in the short term.

2 Problem Statement

The ecological and economic cost induced by the lack of information system control is currently not taken into account in the steering tools of organizations' “responsible digital” policies.

This consideration, for integration within “responsible digital” roadmaps, requires having:

1. an evaluation methodology;
2. appropriate indicators, compliant with “SMART” principles, *i.e.*: Specific, Measurable, Acceptable, Realistic, and Time-bound; and
3. means of action within the organization.

This document aims to propose a methodology concerning the above points 1 and 2, allowing contribution to the inclusion of information system control criteria within the steering tools of a responsible digital roadmap. It proposes a conceptual framework, an analysis methodology, and the definition of individual, collective, and global indicators for calculating control scores capable of guiding an action plan.

3 Concepts and Terminology

Dependency exists between two components if replacing one with a third-party component requires replacing the other to ensure system operation. This is the case for software designed for a given operating system, in the absence of emulators. This is a major pressure lever on users within captive markets.

Adherence exists if replacing one component leads to a degradation in the quality of services provided by the other. Such is the case of a shared calendar that only offers a plugin with certain video conferencing systems, to the exclusion of other systems.

Compatibility exists when several products can work together without users being able to know how the products interact (closed data formats and protocols) or being able to intervene to replace one of the products with one of their choice. Compatibility results from a contractual agreement between product designers, which excludes users.

Interoperability exists when several products can work together and the modes of their interaction are known, thanks to the existence of an open communication format or protocol. These specifications are open when they are public and freely reusable by anyone (such as the PDF format), and free when their definition and evolution are subject to a governance process open to all interested stakeholders.

A software license is considered a **free/libre software license** if it simultaneously guarantees the four freedoms defined by the Free Software Foundation:

- freedom to use the obtained software;
- freedom to copy the obtained software;
- freedom to modify the obtained software (which requires access to the source code);
- freedom to redistribute the modified software (this is not an obligation).

Non-free licenses are licenses that do not respect at least one of the four freedoms. They will be said to be more or less **open** depending on the degree of freedoms they grant. They cannot, however, be qualified as “**open source**”, this term being relative to the Open Source Initiative's ten-criteria Open Source Definition, and matching in its effects the definition of a free/libre license.

Free/libre software is software for which at least one distribution channel accessible to the users considered uses a free license (without the software necessarily being freely accessible to the public under this license, as for example in the case of a closed consortium).

Community refers to a pool of human skills attached to a given good or service, such as a user community, support community, development community, etc.

Resilience is the ability of a system, such as software, to survive the disappearance of its current publisher (company or community). Simple access to source code is not a sufficient guarantee, as the ability for third parties to take over the software's development is conditioned by the rights granted by the license. The resilience of closed-source software is nil, except in the case of provisions taken by the publisher for the escrow of its intangible assets, with transfer to customers of source codes and related rights in case of failure.

Software resilience requires the existence of people capable of taking over its development. Free software without a community is much less resilient than software with a vibrant community of contributors.

4 Evaluation Methodology

In order to be easily understood and adopted, the proposed evaluation method must not lead to collecting and processing excessive amounts of data.

4.1 Implementation Modalities

Data collection must be able to be pooled with a preexisting collection, whose data is useful for implementing the evaluation method. In this case, it is the mapping of the information system, inventorying the hardware and software fleet used in the organization. The objective of information system control therefore constitutes an additional motivation for building this mapping. It is enriched with specific information, constituting a graph of interdependencies between hardware, services, and software.

Indicators must be simple to understand. The results provided must be easily interpretable. In particular, an aggregate presentation must give a synthetic view of the subjects to be treated as a priority. Indicators must be sufficiently sensitive so that a single relevant action on the information system leads to an improvement in the result, thus making the actions undertaken perceptible.

Each component of the information system must be evaluated in isolation, based on the characteristics specific to its supplier and ecosystem, but also in relation to other components with which it interacts, in order to identify dependencies and adherences.

4.2 Dependency Scores

The dependency scores calculated are therefore of three types:

- individual, at the level of each identified hardware and software component;
- collective, at the level of each interdependent set identified through the dependency graph from the information system mapping;
- global, thus providing a Key Performance Indicator (KPI) to decision-making levels.

For mathematical reasons (effect of the multiplication operation described below), the calculated scores are risk scores, not compliance scores: the higher the scores, the less the information system is controlled; the lower they are (close to zero, in practice), the more the information system is controlled.

4.3 Desirable Properties

The calculation method must exhibit the following properties:

- The more components an interdependency mobilizes, the riskier it is and the higher its score must be. The collective score of a technology must therefore depend on the size of the aggregates formed by the interconnected subsets of the dependency graph;
- Some aggregates are more critical than others. For example, for an organization whose core business is not publishing, the graphics production software suite, even if made up of highly

interdependent tools, should not represent the first project. A criticality multiplier coefficient must therefore be associated with each aggregate, to weight its score;

- Improving the control on a non-critical function should not systematically lead to a decrease in the global score, as this could give the impression that the control problem is being resolved, when the essential points have not been addressed. The global score cannot therefore be the sum of the aggregate scores, but rather the maximum of the weighted aggregate scores. Thus, the score will not decrease until the most critical aggregate has been addressed, either by reducing its size (removing an item from the aggregate, or reducing its criticality through the implementation of security measures).

4.4 Component Identification

Let C be the set of information system components. These can be either hardware or software.

In the case of active hardware embedding software (network equipment firmware, etc.), the equipment will be considered in the form of two distinct components, likely dependent on each other. This will result in two scores being calculated: one for the hardware part and one for the software part.

Similarly, in the case of client-server services, the client part and the server part will be considered as two distinct components (or multi-component systems). This will allow separately addressing dependency issues with respect to the server and the client.

4.5 Dependency Graph

The dependency graph G of the information system is constructed as follows. The vertices of G represent the components C_i of the system, and there exists in G an edge between two components C_i and C_j (i.e., C_i is connected to C_j , without any link orientation) if C_i depends on C_j , or vice versa.

Dependency criteria depend on the type of components. They are most often functional dependencies, such as, for example, the specificity of software with respect to a given type of operating system, the existence of compatibility without interoperability between two software in a chain (notably through the use of closed file formats), etc.

Adherences do not constitute dependencies as such. However, strong adherences, which heavily penalize the efficiency and quality of people's work, can be integrated into the dependency graph to represent a specific objective to be addressed.

4.6 Individual Score (Hardware)

The individual score of a hardware component is calculated by summing the values associated with five criteria, defined in the table below. When the criterion is not applicable, a score of zero must be applied to the corresponding criterion, so that it does not penalize the score.

Criterion	Value
-----------	-------

	0	1	2	3
Warranty duration	Very long	Long	Short	Very short
Spare parts supply duration (primary source)	Very long	Long	Short	Very short
Spare parts supply duration (secondary sources / reuse)	Very long	Long	Short	Very short
Reprogrammable	Yes, freely		Only by manufacturer-supplied software	No
Connectors*	Standardized, with converters**	Standardized, without converters	Shared by several manufacturers	Proper to manufacturer

* The Connectors score will be the maximum of the scores established for each connector the device has, both input and output.

** Converters are auxiliary devices allowing preservation of interoperability with older technologies, such as HDMI with respect to VGA, or USB-C with respect to USB-B.

The individual score of a hardware component is therefore between 0 (total control of the component) and 15 (uncontrolled component).

4.7 Individual Score (Software)

The individual score of a software component is calculated by summing the values associated with seven criteria, defined in the table below. When the criterion is not applicable, a score of zero must be applied to the corresponding criterion, so that it does not penalize the score.

Criterion	Value			
	0	1	2	3
Support	24 / 7	Guaranteed	Partial	None
License	Free/Libre	Allows for local modification		Completely closed
Source code	Available along with production tools	Available without production tools	In escrow	Hidden
Community	Very large and very active, support possible	Large and active, listening possible	Several identifiable contributors	None
Input data format	Open, free governance	Open, publisher governance	Closed, with converters	Closed

Internal format	data	Open, governance	free	Open, publisher governance	Closed, converters	with	Closed
Output format	data	Open, governance	free	Open, publisher governance	Closed, converters	with	Closed

The individual score of a software component is therefore between 0 (total control of the component) and 21 (uncontrolled component).

4.8 Calculation Formulas

Let C_i be a hardware or software component. The individual score $S(C_i)$ of each component is calculated based on the tables presented in the previous sections.

The aggregates A_j are constructed as follows: the aggregate of a component consists of the component and the set of other components to which it is directly connected in the dependency graph. A single component can therefore be present in several aggregates. Consequently, components independent of all others are placed in “individual” aggregates that contain only themselves.

The dependency score of the aggregate A_j is the sum of the scores of all components in the aggregate. It is given by the following formula: $S(A_j) = \sum_{C_i \in A_j} S(C_i)$

Each aggregate is weighted by a criticality coefficient, between 1 and 3 (a value of 0 indicates that the aggregate is outside the scope of the study), according to the following table. The criticality coefficient of an aggregate A_j is denoted $c(A_j)$.

Criterion	Value			
	0	1	2	3
Weighting	Outside of perimeter	Non-critical	Moderately critical	Critical

The global dependency score $S(G)$ of the organization, *i.e.*, the score calculated on the entire dependency graph G , is then calculated according to the following formula:

$$S(G) = \max_{A_j \in G} (c(A_j) \times S(A_j))$$

4.9 Interpretation

The use of the “max” operator and of weighting by aggregate size allows one to define achievable objectives. This formula encourages tackling the most massive aggregates first to break them up, or the most critical ones to reduce their criticality, which allows lowering the score of the aggregate in question.

However, the use of this “max” leads to masking the “control debt”, *i.e.*, all the efforts necessary to lower the scores of all aggregates to the lowest possible value. The control debt can be expressed as the sum of the aggregate scores (including the individual scores of non-aggregated components):

$$D(G) = \sum_{A_j \in G} \left(w(A_j) \times S(A_j) \right).$$

5 Conclusion

This document presents a method allowing for the inclusion of information system control in the definition of a digital roadmap. This method aims for simplicity, to be easily implementable at low cost, and for operationality, so that the indicators it proposes can be translated into understandable and achievable objectives.

This document may be freely copied and distributed provided that its content is not modified in any way, and in particular that the author's name continues to appear, as well as this text.

