Bridging the Gap Between Certification and Software Development

Claudio A. Ardagna
Università degli Studi di Milano
Department of Computer Science
Milan, Italy
claudio.ardagna@unimi.it

Nicola Bena
Università degli Studi di Milano
Department of Computer Science
Milan, Italy
nicola.bena@unimi.it

Ramon Martín de Pozuelo
CaixaBank
Security Innovation & Transformation
Barcelona, Spain
rmartindepozuelo@caixabank.com

ABSTRACT

While certification is widely recognized as a means to increase system trustworthiness and reduce uncertainty in decision making, it faces severe challenges preventing a wider adoption thereof. Certification is not adequately planned and integrated within the development process, leading to suboptimal scenarios where certification introduces the need to further modify the developed system with high costs. We propose a methodology that bridges the gap between software development and certification processes. Our methodology automatically produces the certification requirements driving all steps of the development process, and maximizes the strength of certificates while taking costs under control. We formalize the above problem as a multi-objective mathematical program and solve it through a genetic algorithm. The proposed approach is tested in a real-world, cloud-based financial scenario at CaixaBank and its performance and quality is evaluated in a simulated scenario.

CCS CONCEPTS

• **Security and privacy** → *Security requirements*; *Software security engineering*; • **Software and its engineering** → *Software design techniques*; *Software design tradeoffs*; *Software design engineering*; *Empirical software validation.*

KEYWORDS

Certification, Software Development, Security

ACM Reference Format:
Claudio A. Ardagna, Nicola Bena, and Ramon Martín de Pozuelo. 2022. Bridging the Gap Between Certification and Software Development. In *Proceedings of ARES 2022.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3538969.3539012

1 INTRODUCTION

We live in a globally interconnected society characterized by pervasive ubiquitous devices and communication technologies. ICT systems, from small-scale smart homes to large-scale smart cities, govern many aspects of our life, with services such as smart healthcare, smart transportation, and smart grid improving the efficiency and effectiveness of daily activities. The price we pay for such convenience is an increased risk for users' security, safety, and privacy, up to scenarios where users' life can be endangered by malfunctioning or malicious attacks. While this environment and the risks it entails are known and, at least partially, addressed, many users are still recalcitrant towards this smart revolution.

The main reason behind this negative attitude lies in the lack of (perceived) trustworthiness and transparency of these systems, which are often used as black-boxes with no clear evidence on their behavior. The research and the industrial communities, as well as international regulators, are increasingly focusing on solutions aimed to boost system trustworthiness. In this context, certification is becoming a preferred solution to prove whether the system holds a set of certification requirements (e.g., reliability and integrity), and, in turn, increase the confidence on its correct behavior and trustworthiness [5]. Despite all efforts, the adoption of certification is still limited, especially in modern (distributed) systems where software is developed as a service and deployed on cloud-edge infrastructures. This mainly relates to the difficult match between current development processes and existing certification methodologies, introducing high costs and making certification a time-consuming, error-prone, and suboptimal process, which is not easily applicable to the above modern systems. On one side, certification is typically a rigid process, whose costs need to be carefully evaluated [13, 40, 44] in a world where security budgets are often small [45]. On the other side, the development team cannot cope with modern development practices, heavy-regulated environments, and certification methodologies due to their intrinsic mismatches [31, 44]. As a consequence, especially in domains where certification is optional, only a fraction of traditional software is statically certified [14], while nearly no modern systems come with a certificate proving their non-functional properties [5].

Our approach aims to fill in the above gaps, departing from the traditional assumption of considering development and certification processes as loosely-coupled, often sequential, activities. We rather target a development process that is driven by and adapts to certification requirements since system design, producing a certification-ready system. Our approach supports the development team in automatically retrieving the certification requirements in terms of properties to be certified such that *i)* property strength in certificates, and in turn system strength, are maximized, and

ARES'22, August 23–26, 2022, Vienna, Austria

ACM ISBN 978-1-4503-9670-7/22/08…$15.00
https://doi.org/10.1145/3538969.3539012

ii) costs are kept under control. It shifts certification *to the left*, supporting the development team in tightly integrating development and certification processes in both traditional and modern systems.

The contribution of this paper is threefold. We first model the strength and the costs of properties to be certified by enriching the traditional notion of non-functional properties (Section 3). We then formalize the problem of maximizing the strength while matching a given budget, and prove it to be $\mathcal{NP}$*-hard*. We finally propose a heuristic based on genetic algorithm NSGA-III (Section 4), which solves our problem retrieving a small and meaningful set of candidate solutions. The proposed approach is tested in a real-world, cloud-based financial scenario at CaixaBank, a major financial company worldwide, and its performance and quality is evaluated (Section 6) in a simulated scenario.

2 MOTIVATION AND REFERENCE SCENARIO

2.1 Motivation and State of the Art

Certification is an accepted practice especially in safety-critical domains, such as avionics and medical, where malfunctions may threat users' life, and the development process must follow strict guidelines [26, 41]. Certification practices in the industry involve manual activities and documentation, and are mainly based on the *Common Criteria* (CC) standard [25]. They are not applicable in modern systems, including cloud and IoT, due to the intrinsic dynamicity of such environments and the peculiarities of the corresponding development practices requiring many integration and deployment steps in small windows of time. Alternatives do exist (e.g., [5, 35]) and are being developed (e.g., *EUCC* [19], *EUCS* [20], *5G Certification* [21], promoted by the European Union Agency for Cybersecurity), but are still rarely applied.

In general, a *certification process* is a rigorous process that aims to certify a set of *requirements* on a *target of certification* (a software, a service, or a system in the broader sense), by collecting *evidence* from *evidence collection points* implemented on the target. Evidence is collected according to manual documentation, testing, monitoring, and formal methods [5, 22, 25, 37]. Requirements are modeled in different ways according to the considered domain, such as: *i) fixed properties*, modeling generic requirements such as those forming the CIA triad (*Confidentiality*, *Integrity*, *Availability*) [7, 36]; *ii) assurance cases*, modeling requirements as high-level assurance objectives (cases), for which developers have to (manually) prepare supporting evidence [12, 24, 29]; *iii) non-functional properties*, modeling requirements as a fixed property (here referred to as abstract property) refined with specific attributes (e.g., the encryption algorithm for data confidentiality). Non-functional properties mimics *Protection Profiles* of CC [25]. They have been mostly considered in the context of service certification [2] and later enhanced in the context of cloud certification [3, 34, 43], cloud plan adaption [6], DevOps pipelines [4], and to complement and validate risk management [1].

Figure 1(a) presents an overview of the relation between a state-of-the-art certification process and the development process. The certification process is decoupled from the development process and applied a posteriori, when the target of certification is fully designed and implemented. It is a complex and resource-demanding task, adding a significant burden on the development team, including

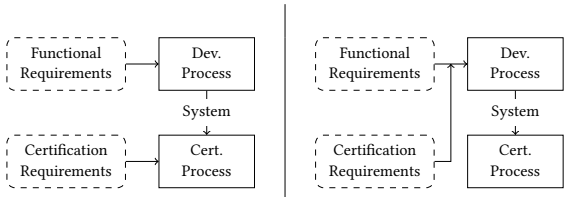

Figure 1: Certification and development processes

software engineers, developers, and architects. The reason lies in two assumptions that virtually all certification approaches made.

Assumption 1: certification requirements (from non-functional properties [3] to assurance cases [12, 29]) are defined at design time before system development starts. This assumption does not hold in real scenarios, since requirements are often not considered at design time or provided by developers without a rigorous process.

Assumption 2: evidence collection points (from written documents [25] to cloud APIs [3], monitoring infrastructures [34], and Trusted Platform Modules [7]) are implemented in advance and are correct. This assumption does not hold in real scenarios, since evidence collection points might not be always available or might be inadequate to support a certification process.

These assumptions are the main hurdles towards fully exploiting certification and limit its adoption in modern systems. The software development process happens well in advance with respect to the certification process or they are just informally bound on the basis of development team's choices. If certification is not planned adequately, it could require more effort than development [22]. Certification requirements are clarified at certification time only, or the system to be certified could not include the correct evidence collection points, requiring *i)* a huge effort in adapting certification requirements to the existing system or vice versa; *ii)* large modifications to the system to support evidence collection points. Costs could then quickly go out of control, resulting in companies avoiding system certification [44]. This is critical in static certification approaches of traditional software systems like Common Criteria, and is exacerbated in modern systems where code changes and releases happen at high rates, with low budgets allocated for security practices [45]. As a result, virtually no modern systems are certified according to industrial standards [14].

Our methodology in Figure 1(b) bridges the gap between software development and certification, supporting developers in the implementation of certification-ready systems. It considers certification and its impact on development earlier, guiding the realization of a system where *i)* certification requirements are clear from the beginning and drive the development of a system supporting them; *ii)* evidence collection points are planned and implemented during the development process. Following the latest research on certification [2, 3], our methodology models a certification requirement as a non-functional property defined as follows.

Definition 2.1. A non-functional property is a pair $p=(\hat{p}, \{(a_i.v_j)\})$ where *i)* $\hat{p}$ is the abstract property (e.g, confidentiality, integrity,

availability); *ii)* $\{(a_i.v_j)\}$ is the set of attribute values refining the property, with a_i the name of attribute and $v_j \in V_i$ the value of attribute a_i.

We note that $V_i=\{v_j\}$ is the set of possible values for attribute a_i. Each property includes attributes values referred to *i)* the property itself and its implementation (e.g., RAID type for low-level data integrity); *ii)* the software development process (e.g., programming language), including deployment and operational details; *iii)* the software certification process, including evidence collection points.

2.2 Reference Scenario

Our reference scenario considers financial services at CaixaBank, a major financial company worldwide. The development team at CaixaBank is currently challenged with development and operations of such services following a DevSecOps approach, often lacking adequate tools and training when development meets security assurance and trustworthiness verification. This problem is particularly critical in this sector, due to *i)* the proliferation of regulations such as PCI-DSS, GDPR, or the upcoming EU Cybersecurity Act, asking to show compliance with a large set of security, privacy, and legal requirements; *ii)* the barriers posed by regulations at point *i)* in integrating new services while assessing and enforcing security and privacy requirements; *iii)* the increasing trend in migrating existing services to the cloud, introducing the need of adapting the development and assessment processes to the target cloud infrastructure. For these reasons, techniques supporting the development and the certification of financial services become fundamental to provide high-quality assurance while keeping costs under control. These techniques drive developers in implementing certification-ready services and increase the perceived trustworthiness of the companies in the medium term [3, 5].

Our reference scenario focuses on the payment service at CaixaBank, providing all functionalities for digital payments and financial information management. CaixaBank wants to migrate its service to the cloud and certify it against PCI-DSS. PCI-DSS is a major standard to which all operators involved in cardholder data processing must comply imposing, in short, to securely process such data, from storage to transmission. The cloud migration requires to re-architect the service and implement new functionalities to benefit from the cloud and, at the same time, to make the certification process faster and cheaper. The development team at CaixaBank is currently integrating the payment service with a PCI-DSS-compliant cloud infrastructure (e.g., Amazon AWS) and developing all components at the basis of the payment service certification.

In the following, we discuss how our methodology can support the development team in implementing a certification-ready service minimizing costs and increasing the robustness of the certified product [39]. To this aim, we focus on PCI-DSS requirement *protect cardholder data*, which includes properties confidentiality p_c, integrity p_i, and availability p_a.

3 CERTIFICATION-DRIVEN DEVELOPMENT

Our methodology supports the development team in finding the best non-functional properties that balance the system strength and the additional cost introduced on the development process in terms of developer knowledge, certification endpoints, and additional functionalities to be included. The development process is driven by the collected properties and results in a certification-ready system with high quality and costs under control. A certification-ready system means that evidence collection points are integrated since the design phase, allowing the certification process to start right after the software development process is completed and retrieve the evidence needed for system certification.

Our methodology is modeled as an optimization problem (Section 3.3) that builds on three main components. The first component is the set of non-functional properties (Definition 2.1). Each property can be initially defined at different levels of granularity, ranging from an abstract property $\hat{p}$ to a fully-specified property $p=(\hat{p}, \{(a_i.v_j)\})$. We note that often the development team defines a property where part of the attributes values are left empty, depending on its initial knowledge. Fully-specified properties $\{p\}$ are the output of the optimization problem and aim to balance the other two components of the optimization problem: *i)* property strength (Section 3.1) and *ii)* cost (Section 3.2).

3.1 Property Strength

Property strength models the robustness of a property verified on a system. For instance, let us consider property integrity. The more the strength of the property, the higher the integrity of the system. We note that each attribute value can contribute to the strength of the corresponding property and must be carefully evaluated. For instance, the cryptographic algorithm used for signature and the key length have an impact on the property strength.

Formally, let $\hat{p}_k$ be an abstract property, and (a_i, V_i) an attribute. The set V_i of attribute values is partitioned in disjoint equivalence classes $c_1, \ldots, c_n$ for $\hat{p}_k$, s.t. *i)* attribute values in class c_i equally contribute to the property strength; *ii)* the contribution of attribute values in class c_i to the property strength is *higher* than the contribution of values in c_{i-1} and *lower* than the contribution of values in c_{i+1}. We then define the contribution of $a_i.v_j$ to $\hat{p}_k$, that is, the contribution to the strength of $\hat{p}_k$ when attribute (a_i, V_i) has value v_j, as $g_k(a_i.v_j)=\frac{j}{n}$, where $j \in \{1, \ldots, n\}$ is the sorted index of the corresponding equivalence class, and n the number of classes. We note that equivalence classes are typically defined by experts. The contribution is 0 when attribute values do not contribute to the property strength.

Example 3.1. Following our reference scenario in Section 2.2, we consider an excerpt of property confidentiality of the data stored by payment service as $p_c=(\hat{p}_c, \{$(standard, *PCI-DSS*), (dev-type, *DevSecOps*), (storage, *AWS-RDS-ENC*), (key-length, *256bit*)$\})$. p_c requires the payment service to show *PCI-DSS* compliance, be developed according to a *DevSecOps* methodology, and integrate a managed relational database *RDS* on AWS encrypting data at rest (*ENC*) with key length *256bit*. Attribute *storage* includes the following attribute values, grouped according to their equivalence classes for p_c: $V_{\text{storage}}=\{\{$*Custom-KV*$\}, \{$*Custom-DBMS*$\}, \{$*AWS-RDS*, *AWS-KV*$\}, \{$*AWS-RDS-ENC*, *AWS-KV-ENC*$\}\}$. "*Custom-*" refers to unmanaged databases, while "*KV-*" to key-value stores and non-relational databases. The contribution of the above attribute values is therefore $\{0.25, 0.5, 0.75, 1\}$ (i.e., 0.25 is the contribution of attribute values in the first equivalence class, and so on).

The strength of a property p_k, denoted as $\lambda(p_k)$, is the sum of the contribution of the composing attribute values, as follows.

$$\lambda(p_k) = \sum_{a_i.v_j \in p_k} g_k(a_i.v_j) \tag{1}$$

3.2 Costs

Each choice made on the attributes of a given property p_k has an impact on the effort needed to develop a system supporting p_k. We therefore associate each attribute value with a monetary *cost*, denoted by function $c\colon V_i \to \mathbb{N}$, and define the *cost* of a non-functional property p_k as a summation over the costs of the attribute values composing it, as follows.

$$\sum_{a_i.v_j \in p_k} c(a_i.v_j), \tag{2}$$

We note that the total cost introduced by the properties must be within the budget available for the development process, which is denoted as $\mathcal{B}$. The cost of attribute values models the additional developer knowledge to be formed/recruited (e.g., when a specific programming language is selected), and the additional effort to develop certification endpoints (e.g., preparing evidence collection points for the certification process) and additional functionalities (e.g., the integration of a new specific encryption algorithm). It also models costs introduced by new services or infrastructures supporting system deployment. Some of these costs can be estimated by models such as COCOMO 2.0 [11]. We note that our methodology is independent of the solution adopted for cost estimation, which is outside the scope of this paper.

Example 3.2. Following Example 3.1, we estimate the cost of certifying property p_c as the costs of the modifications to perform on the payment service. The attribute values impacting the most on p_c are as follows: *i)* (dev-type, *DevSecOps*) as the cost of an external *DevSecOps* expert responsible for the payment service migration; *ii)* (storage, *AWS-RDS-ENC*) as the cost of an AWS-hosted instances of the database, including the eventual cost to certify this additional component. In this case, there are no additional costs beyond the instance itself, since AWS services are already certified for PCI-DSS compliance.

3.3 Problem Statement

Our methodology is modeled as an optimization problem that receives as input the budget and the set of (abstract) properties in Definition 2.1 with the corresponding attributes, contributions, and costs. It returns as output a set of (fully-specified) non-functional properties whose strength is maximized and cost is below the budget. We aim to optimize (and later certify) the properties globally; optimizing (and certifying) each property individually would in fact result in conflicting certification requirements and impair the certification process in the release of a certificate for each property. Each property is composed of peculiar attributes whose value is different (possibly not present) in other properties and shared attributes whose value is the same in each property. Peculiar attributes mostly refer to property-specific configurations, such as *key-length* with value *256bit* for property p_c. Shared attributes mostly refer to the development process. For instance, p_c, p_i, and p_a have the following shared attributes: *standard*, *dev-type*, and *storage*. We denote A as the union of peculiar and shared attributes, with the latters added only once.

Problem 1 (Max-Property). *Given a set of abstract properties $\{\hat{p}\}$ s.t. $|\{\hat{p}\}|$=K taking values in A, find the best attribute values for A for each property $\in\{\hat{p}\}$ s.t. properties strength $\{\lambda(p)\}$ is maximized and budget $\mathcal{B}$ is honored.*

In other words, Max-Property aims to find a set of fully-specified non-functional properties refining the abstract properties while maximizing property strength and addressing the budget. Max-Property can be generalized to a problem where the properties are partially specified and the optimization must find the best values for unspecified attributes.

Theorem 3.3. *The Max-Property is $\mathcal{NP}$-hard.*

Proof. The proof is a reduction from the *$\mathcal{NP}$-hard* problem of the *multi-objective multiple-choice knapsack problem*, formulated as follows: *given a set of disjoint classes of items, each item j of class i having a weight and K values (i.e., profits), pack exactly one item per class in a knapsack of capacity C without violating it and maximizing the sum of the corresponding values for $k = 1, \ldots, K$* [18, 32]. The correspondence between the Max-Property and the aforementioned knapsack problem is as follows. Each class i translates to an attribute $(a_i, V_i)\in A$; each item j of class i translates to an attribute value $a_i.v_j$; each value k of item j of class i translates to the contribution of the corresponding attribute value to the strength of the k-th non-functional property (i.e., $g_{p_k}(a_i.v_j)$); the weight of item j of class i translates to the monetary cost of the corresponding attribute (i.e., $c(a_i.v_j)$); knapsack capacity C translates to monetary budget $\mathcal{B}$. □

A binary programming formulation of Max-Property is as follows.

$$\max \sum_{(a_i,V_i)\in A} \sum_{v_j \in V_i} g_k(a_i.v_j)\cdot x_{ij} \quad k = 1,\ldots,K \tag{3a}$$

$$\text{s.t.} \sum_{j}^{|V_i \in A|} x_{ij} = 1 \qquad i = 1,\ldots,|A| \tag{3b}$$

$$\sum_{(a_i,V_i)\in A} \sum_{v_j \in V i} c(a_i.v_j)\cdot x_{ij} \le \mathcal{B} \tag{3c}$$

$$x_{ij} \in \{0,1\} \qquad i = 1,\ldots,|A|, j = 1,\ldots,|V_i| \tag{3d}$$

where *i)* equation (3a) presents the K objective functions, one for each property, making Max-Property a multi-objective problem. Each objective function models the maximization of the corresponding property strength according to the contribution of the chosen attribute values; *ii)* equation (3b) requires to choose one value for each attribute in A; *iii)* equation (3c) defines the requirement on the monetary budget; *iv)* equation (3d) presents the binary decision variables x_{ij}, whose value is 1 when the j-th attribute value of the i-th attribute is chosen. We note that a valid solution x to Max-Property is a set $\{p_1, \ldots, p_K\}$ of K non-functional properties, with p_i being a refinement of the corresponding abstract property $\hat{p}_i$ given as input and K the number of abstract properties.

4 HEURISTIC SOLUTION

Multi-objective optimization problems lack the definition of *optimal* solution, because, in general, there are no solutions simultaneously optimizing all objectives. The resolution process then passes from the definition of dominance [33], as follows.

Definition 4.1. Let x, x' be two feasible solutions and $\lambda(p_k)$ ($\lambda(p'_k)$, resp.) the strength of $p_k \in x$ ($p'_k \in x'$, resp.). x' dominates x iff

i) $\forall k \in \{1, \ldots, K\}, \lambda(p'_k) \geq \lambda(p_k)$;
ii) $\exists k \in \{1, \ldots, K\}, \lambda(p'_k) > \lambda(p_k)$.

Consequently, x' is *nondominated* (or *Pareto optimal*) iff it does not exist any other solution x dominating x'.

In other words, a solution x' dominates another x if, for each non-functional property p'_k therein, the corresponding strength $\lambda(p'_k)$ is never worse, and strictly better at least once. The set of all nondominated solutions is called *Pareto set*, denoted as $S(X)$, and the corresponding image is called *Pareto front*, denoted as $S(Y)$. These sets have typically (very) large cardinality [38].

Max-Property is $\mathcal{NP}$*-hard* and cannot be solved to optimality, that is, finding the complete and exact sets $S(X)$ and $S(Y)$. Our approach *i)* computes an approximation of the above sets (Section 4.1); *ii)* selects a meaningful solution from such sets (Section 4.2). For brevity, in the following, we only consider $S(Y)$, whose retrieved approximation is denoted as $\widetilde{S}(Y)$.

4.1 Finding the Nondominated Solutions

We use NSGA-III [17, 30], a genetic algorithm designed for solving complex optimization problems with many objectives (> 3), to compute $\widetilde{S}(Y)$. A set of *encoded* solutions, called *population*, evolves during the algorithm iterations, until a stopping condition is met, as follows.

Input. The algorithm receives as input the set of attribute values A for abstract properties $\{\hat{p}_1, \ldots, \hat{p}_K\}$, each attribute value with the corresponding contribution to the property strength and cost (Section 3). It also receives as input a set of *reference points*, that is, specific configurations of non-functional properties to guide the evolutionary search. They are generated uniformly to let the algorithm explore the whole search space with equal probability.

Encoding and Initial Population. The algorithm randomly generates the initial population. Random non-functional properties are extracted according to the provided attributes, without considering their costs wrt the total budget (*feasibility*). Each population member represents a set of non-functional properties, encoded as an integer vector. The i-th attribute is encoded at position i, taking value $j \in [0, |V_i|]$ representing the value $v_j \in V_i$ of attribute a_i. The vector length is $|A|$ (Section 3.3). The population size equals the number of reference points, denoted as N.

Offsprings Generation. At each iteration, the algorithm applies selection, crossover, and mutation to the current population to generate offsprings (i.e., a new population) of the same size. Selection extracts pairs of parents. It selects each parent out of two random candidates from the current population, preferring feasibility over infeasibility, and smaller constraint violation values over larger ones (*binary tournament selection* [15]). Crossover combines each pair of parents to generate a new pair of offsprings. For each offspring, it *combines* the parents according to a probability distribution (*simulated binary crossover* [17]). Mutation modifies offsprings according to the same probability distribution of crossover (*polynomial mutation* [16]). Finally, a repair operator ensures that the generated offsprings respect the encoding.

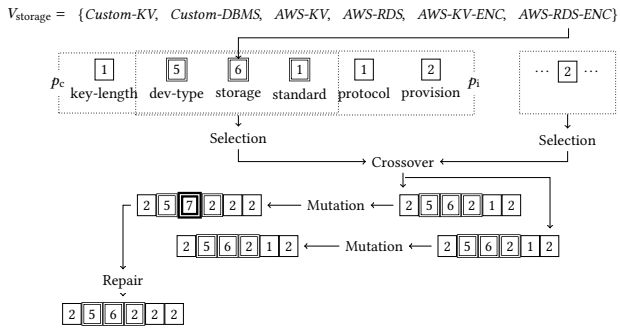

Figure 2: Overview of NSGA-III

Population Generation. At each iteration, the algorithm processes current population and offsprings to generate the population for the next iteration. It extracts a set (called *front*) from the union of current population and offsprings according to a constraint-domination relationship [30] s.t., for any pair of solutions in the combined population, *i)* a feasible solution is chosen over an infeasible one; *ii)* an infeasible solution with the smallest constraint violation value is chosen over another infeasible one; *iii)* a feasible nondominated solution is chosen over another feasible one. Once a first front is filled up with all the members satisfying the requirements, the procedure is repeated over and over with the remaining members, until they are all assigned to a front. Fronts are then associated with a rank: the first created front is associated with 1, the second with 2, and so on. Population for the next iteration is built by taking N members from the fronts in ascending rank order. This way, feasible solutions are chosen first, followed by infeasible solutions with the smaller constraint violation values.

Output. The algorithm returns the first-ranked front in the last generated population. Recalling that a population member encodes a possible configuration for all the K non-functional properties, each population member is a potential, approximated solution of Max-Property. We note that some members might be infeasible, as they are approximation. Nevertheless, among all the computed infeasible non-functional properties (if any), the algorithm favors non-functional properties violating the least the given budget.

Example 4.2. Following Example 3.2, Figure 2 shows an iteration of NSGA-III, reporting one population member with attributes of p_c and p_i. Dotted boxes group attributes in properties. Each solid square indicates an attribute (single line for peculiar attributes, double line for shared ones), whose content is the position of the value in the set of attribute values. For instance, attribute *storage* has value 6 and refers to *AWS-RDS-ENC*, which is at 6-th position in the set V_{storage}. Selection selects pairs of parents used in crossover to generate pairs of offsprings. Each offspring is then mutated to obtain

modified attribute values. For instance, offspring $\{2, 5, 6, 2, 1, 2\}$ is mutated obtaining $\{2, 5, 7, 2, 2, 2\}$. These operations produce an invalid offspring, since *storage* has value 7 while the largest possible value is 6, requiring repair to $\{2, 5, 6, 2, 2, 2\}$. Offsprings and current population are then merged to generate the new population.

4.2 Selecting Significant Solutions

Let P be the set of solutions returned by NSGA-III, $p_{k,l}$ the k-th property encoded in the l-th population member. Our approach returns the best solution according to one of the following metrics. We note that these metrics are based on the strength or costs of the properties in P (i.e., operate in the codomain), but return the corresponding properties in P (i.e., the domain).

Maximum Strength. It proposes the solution maximizing the cumulative strength, according to the following equation.

$$\arg \max_{l=1,\ldots,|P|} \sum_{k=1}^{K} \lambda(p_{k,l}) \tag{4}$$

Minimum Cost. It proposes the solution minimizing the cumulative monetary cost, according to the following equation

$$\arg \min_{l=1,\ldots,|P|} \sum_{a_i.v_j \in A_l} c(a_i.v_j) \tag{5}$$

where A_l is the set of attribute values across all properties in the l-th population member.

Minimum Penalty. It proposes the solution with minimum highest difference (i.e., penalty) from an *utopian solution*. The utopian solution λ^{++} is retrieved by adding a small ϵ to the *ideal solution* λ^{+}, which is computed according to the following equation.

$$\lambda_k^{+} = \max_{l=1,\ldots,|P|} \lambda(p_{k,l}) \quad k = 1, \ldots, K \tag{6}$$

where λ_k^{+} (λ_k^{++}, resp.) corresponds to the maximum strength of property k (the strength increased by ϵ, resp.). The minimum penalty solution is retrieved according to the following equation [38].

$$\arg \min_{l=1,\ldots,|P|} \max_{k=1,\ldots,K} \left(w_k \cdot (\lambda_k^{++} - \lambda(p_{k,l})) \right) + \rho \sum_{k=1}^{K} w_k \cdot (\lambda_k^{++} - \lambda(p_{k,l})) \tag{7}$$

For each member l in the population P and for each property strength $\lambda(p_{k,l})$ in it, equation (7) first retrieves the difference between $\lambda(p_{k,l})$ and the utopian solution λ_k^{++}; this difference is called *penalty*. Second, for each solution in the population, it retrieves the highest penalty (i.e., max). Third, it retrieves the solution whose penalty is minimum (i.e., min). Weights w_k favor solutions that are closer to the utopian values [42]. The additive term permits to retrieve a solution that is *ρ-properly Pareto optimal* [38]. Formally, a Pareto optimal solution $x'=\{p'_1, \ldots, p'_K\} \in P$ is ρ-properly Pareto optimal if [38]:

1. it is Pareto optimal (Definition 4.1), and
2. it exists $M=1+1/\rho$ s.t. for each feasible solution $x=\{p_1, \ldots, p_K\} \in P$ and for each property $p_k \in x$ where $\lambda(p_k) > \lambda(p'_k)$, there exists at least one property $p_j \in x$ where $\lambda(p'_j) > \lambda(p_j)$ and

$$\frac{\lambda(p_k) - \lambda(p'_k)}{\lambda(p'_j) - \lambda(p_j)} \leq M \tag{8}$$

A solution is ρ-properly Pareto optimal if there is at least one pair of properties strength (objectives) s.t. an improvement in one property strength is possible only by decreasing the other property strength; ρ bounds this variation [38].

Example 4.3. Following Example 4.2, the development team at CaixaBank chooses the solution minimizing the overall cost. It includes p_c as $p_c=(\hat{p}_c, \{$(standard, *PCI-DSS*), (dev-type, *DevSecOps*), (storage, *AWS-RDS-ENC*),(key-length, *256bit*)$\})$ with strength 3.5. The total cost of the retrieved p_c, p_i, p_a is discussed in Section 5.

5 WALKTHROUGH

We present a walkthrough of our methodology in the reference scenario in Section 2.2. Developers at CaixaBank need to migrate their payment service to the cloud and then certify it according to PCI-DSS standard, in general, and requirement *protect cardholder data*, in particular. Their goal is to extend the payment service in a way that, once migrated, it can be easily certified with minimum effort. This means that the service must support all the requirements imposed by the PCI-DSS standard in advance, including the evidence collection points to collect relevant evidence.

Cost model. For clarity, we consider a simple cost model inspired by the work in COCOMO [11]. The cost of each attribute value is defined using labels in the set {*Very Low*, *Low*, *Medium*, *High*, *Very High*}. Each label is then associated with a cost modeled as the effort in *person-day* (*pd*) requested to the development team to address the specific attribute value, that is, *Very Low*=1pd, *Low*=7pd, *Medium*=15pd, *High*=30pd, *Very High*=60pd. We note that these values represent an average estimation and depend on the considered scenario. The monetary costs are finally calculated assuming an average hourly labour cost of 23€ in Spain.[1] We note that, for simplicity, costs related to the adoption of services and infrastructures have been assumed to be negligible; more sophisticated cost models can be applied in our methodology.

Information collection. Developers translate PCI-DSS requirements in non-functional properties p_c, p_i, and p_a, defining attributes, their possible values, and their costs. We note that cost estimation has been done manually and validated by CaixaBank. Table 1 summarizes this information. For instance, attribute *standard* defines possible standards including *PCI-DSS*, *HIPAA*, and *GDPR*. Attribute *lang* defines possible programming languages with different contributions to the properties' strength: *Go* and *Rust* have high contribution (0.75 and 1, resp.) and high cost (*High* and *Very High*, resp.). The same applies for attribute *dev-type*. Configuration attributes such as *key-length* and *protocol* have different contributions with the same cost; their choice in fact does not add any additional burden. Similar to *storage* in Example 3.1, a managed platform on AWS (either *EKS*, that is, Kubernetes, or *Lambda*) requires less effort than a non-managed one. We note that shared attributes (i.e., attributes referred to more than one property) have equal contribution to any of the corresponding properties. The monetary budget $\mathcal{B}$ is calculated

[1] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Hourly_labour_costs

Table 1: Attributes values of our reference scenario. Selected property attributes are highlighted in bold.

Attr. name	Values	Contrib.	Cost	Properties
standard	***PCI-DSS***	**0.5**	***Very High***	p_c, p_i, p_a
	HIPAA	0.5	*Very High*	
	GDPR	1	*Very High*	
lang	*C*	0.25	*Very High*	p_c, p_i, p_a
	Python	0.5	*Medium*	
	Java	**0.75**	***Low***	
	Go	0.75	*High*	
	Rust	1	*Very High*	
dev-type	*Waterfall*	0.2	*Very High*	p_c, p_i, p_a
	Prototype	0.4	*Very High*	
	Spiral	0.6	*Medium*	
	DevOps	0.8	*High*	
	DevSecOps	**1**	***High***	
storage	*Custom-KV*	0.25	*Very High*	p_c, p_i, p_a
	Custom-DBMS	0.5	*Very High*	
	AWS-KV	0.75	*Low*	
	AWS-RDS	0.75	*Low*	
	AWS-KV-ENC	1	*Low*	
	AWS-RDS-ENC	**1**	***Low***	
key-length	*128bit*	0.5	*Very Low*	p_c
	256bit	**1**	***Very Low***	
protocol	*TLSv1.2*	0.5	*Very Low*	p_i
	TLSv1.3	**1**	***Very Low***	
provision	*manual*	0.5	*High*	p_i
	auto	**1**	***Very Low***	
platform	*none*	0.25	*High*	p_a
	Kube-Custom	0.5	*Very High*	
	AWS-EKS	**0.75**	***Low***	
	AWS-Lambda	1	*Medium*	

assuming a *Medium* cost (15pd) for each attribute, that is, a total of 120pd (i.e., 15pd×8 attributes). According to the average hourly labour cost of 23€ in Spain, the monetary budget $\mathcal{B}$ is 22.080€.

Property optimization. Information collected in Table 1 is fed in our methodology, finding the set of fully-specified, non-functional properties honoring the specified budget, as described in Section 4.1. Developers select metric *Minimum Cost* in Section 4.2, retrieving the fully-specified properties p_c, p_i, p_a minimizing the overall cost. The selected property attributes are highlighted in bold in Table 1 and are as follows: p_c=($\hat{p}_c$, {(key-length, *256bit*)}), p_i=($\hat{p}_i$, {(protocol, *TLSv1.3*), (provision, *auto*)}), p_a=($\hat{p}_a$, {(platform, *AWS-EKS*)}). The solution has shared attributes {(standard, *PCI-DSS*),(language, *Java*), (dev-type, *DevSecOps*), (storage, *AWS-RDS-ENC*)}, with strengths $\lambda(p_c)$= 4.25, $\lambda(p_i)$=5.25, $\lambda(p_a)$=4, and total cost of 20.976€ (i.e., 114 person-day), under the budget of 22.080€.

Development. The development team then starts the migration, adapting the payment service to AWS cloud. This includes modifications on the codebase to support the new managed database (property p_c), the configuration of automatic provision of TLS certificates (property p_i), the deployment on AWS Kubernetes (property p_a). Once the migration is complete, payment service can successfully undergo PCI-DSS certification.

Discussion. By considering certification earlier in the development process, CaixaBank developers can adapt an existing service to a cloud-based certification-ready service minimizing the overall effort. The global optimization of all the properties leads toa solution (i.e., *development plan*) without conflicting requirements, while keeping development and certification costs under control. For instance, following a traditional development process decoupled by the certification process, the developers could integrate an unmanaged database within the VM deploying the payment service. This choice gives an immediate benefit in terms of development costs, while requiring significant costs at certification time, since the custom DBMS as well as the service should be certified against PCI-DSS from scratch. By contrast, following our methodology, the developers opt for a PCI-DSS-compliant AWS cloud infrastructure. This stresses the importance of planning certification from the beginning, building an adequate plan to support developers. Due to the large number of variables involved, our approach automate this planning phase, driving developers in the identification of the best properties according to the selected metric.

Table 2: Settings for NSGA-III

Parameter	Value
Prob. of crossover	1
Prob. distrib. in crossover (η)	30
Prob. of mutation	$1/\|N\|$
Prob. distrib. in mutation (η)	20

(a) main parameters

Parameter	Value
Variation of obj.	0.0025
Variation of constr.	1^{-6}
Max iterations	100
Iterations window	30

(b) stopping conditions

6 EXPERIMENTS

We experimentally evaluated the performance and quality of our approach in a simulated environment. Experiments have been run on a VM featuring 8 vCPU Intel® Xeon Silver 4216 CPU@2.10 GHz, 64 GBs of RAM, operating system Ubuntu 20.04 x64, using Python 3.8.10 with libraries *numpy* [23], and *pymoo* [9].

We configured NSGA-III following best practices in the literature [17, 30] with *i)* non-functional properties varying in 4, 5, 6, 7 and population size (N) in 160, 212, 280, 360, resp. (Table 2(a)), *ii)* stopping conditions considering the average variation in the objective functions, the average variation in the constraint violation (the monetary budget), and the number of generations (iterations) (Table 2(b)); conditions are evaluated every 30 iterations, and the condition reached first stops the algorithm. Reference points have been generated using the method in [10].

6.1 Performance

We compared the performance of our heuristic approach in Section 4.1 with an exhaustive brute force search realized in Python. We varied *i)* the number K of properties in 4, 5, 6, 7; *ii)* the total number of attributes in $i+i*K$, with $i\in\{4, 8, 12\}$, such that there exists i shared attributes among all properties and i peculiar attributes for each property (a total of $i*K$ peculiar attributes); *iii)* the number of attribute values $|V_i|$ for each attribute i in 10, 25, 40. Partitions in equivalence classes (Section 3.1) and costs (Section 3.2) were randomly generated. Each combination of the above parameters corresponds to an instance of problem Max-Property. The number of possible solutions is equal to $\prod |V_i|$ for i=1, . . . , $|A|$, that is, the combinations of all attributes values. The brute force algorithm splits the search space in the number of CPU cores (8) and proceeds

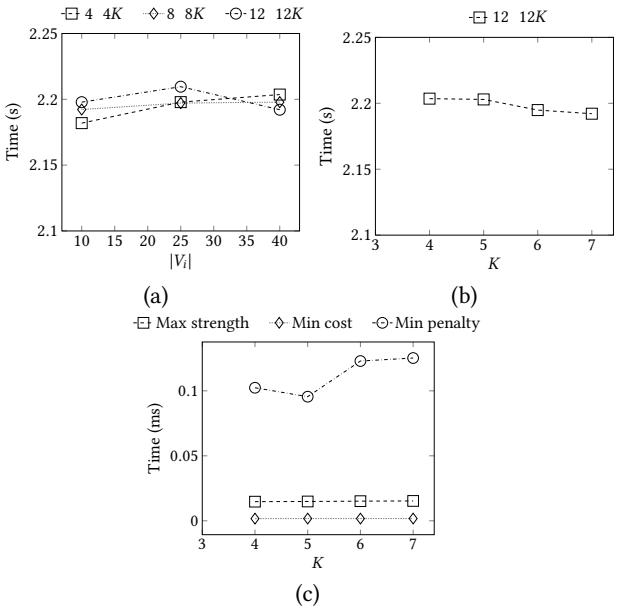

Figure 3: Performance of NSGA-III (a), (b), and selection of relevant solutions (c).

in parallel, exploring all the possible solutions forming the Pareto front $S(Y)$, with a maximum execution time of 15 minutes.

Figure 3(a) shows the performance of NSGA-III with K=7 varying the number of attributes and attributes values. Figure 3(b) shows the performance of NSGA-III varying the number of properties and, in turn, the population size (Table 2(b)), in a worst case with 12 attributes and 40 attribute values. We observe that the performance never exceeds 2.25s in the worst case. Also, the impact of the input size on the performance is negligible thanks to the optimized libraries we used. On the contrary, brute force algorithm never yielded a solution within the 15-minute threshold, exploring a very negligible part of the search space, 1.12×10^{-12}% on average.

Figure 3(c) shows the performance of the selection of the best solution according to the metrics in Section 4.2, in a worst case with 12 attributes and 40 attribute values. We note that the time complexity depends on the number of properties only. As expected, selection is faster when using minimum cost and maximum strength; the metric minimum penalty is the slowest, as it involves more complex computations. Execution time never exceeds ≈0.125ms.

6.2 Quality

We executed NSGA-III 100 times on each generated instance of Max-Property (Section 6.1). We repeated each individual execution 5 times and considered the best according to its *Hypervolume*. For each instance, we retrieved the *accumulated* Pareto front $\widetilde{S}^*(Y)$, by combining the individual $\widetilde{S}(Y)$ of each execution according to the dominance relationship in Definition 4.1. This served as our reference solution. We note that this approach is recommended when exact solutions are not available, like in our case [27]. We then evaluated the quality of our approach by *i)* measuring how much our approximated Pareto front $\widetilde{S}(Y)$ covers the *accumulated* Pareto front

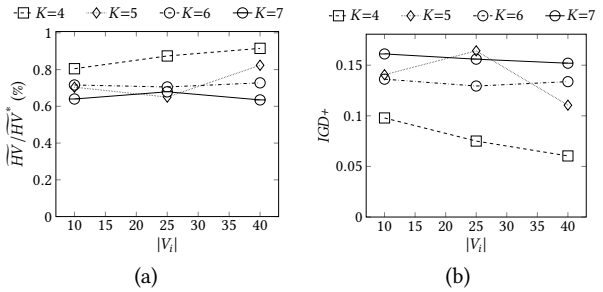

Figure 4: Hypervolume quality (a) and *IGD+* (b).

$\widetilde{S}^*(Y)$; *ii)* comparing, for each instance, the best solution retrieved in each execution and the best among all executions, following the metrics in Section 4.2.

6.2.1 Pareto Front Coverage. We measured the quality of the approximated Pareto front $\widetilde{S}(Y)$ found by NSGA-III using quality indicators *Hypervolume* and *IGD+*.

- *Hypervolume* measures the volume of the area enclosed by $\widetilde{S}(Y)$ and a given reference point far from $\widetilde{S}(Y)$. It is an indicator to be maximized. In our case, we normalized $\widetilde{S}(Y)$ and, for the sake of computation, transformed Max-Property into a minimization problem, where the *ideal* is $(0)^K$ and reference point is $(1.1)^K$. We note that there is no consensus on the choice of the reference point, yet this is the typical approach [33].
- *IGD+* measures the average (modified) Euclidean distance d between each point in $\widetilde{S}^*(Y)$ and the closest point in $\widetilde{S}(Y)$ [28]. It is defined as

$$\text{IGD+}(\widetilde{S}(Y), \widetilde{S}^*(Y)) = \frac{1}{K} \sum_{i=1}^{k} \min_{\widetilde{\lambda}(p_k) \in \widetilde{S}(Y)} d\left(\widetilde{\lambda}^*(p_k), \widetilde{\lambda}(p_k)\right),$$

$$\text{where } d\left(\widetilde{\lambda}^*(p_k), \widetilde{\lambda}(p_k)\right) = \left(\sum_{k=1}^{K} (\max(\widetilde{\lambda}^*(p_k) - \widetilde{\lambda}(p_k), 0))^2\right)^{1/2}.$$

For each execution, we calculated the Hypervolume $\widetilde{HV}$ and computed the quality as $\widetilde{HV}/\widetilde{HV}^*$, where $\widetilde{HV}^*$ is the Hypervolume of $\widetilde{S}^*(Y)$. We then averaged such quality over the 100 executions of each instance. We note that Hypervolume computation is itself $\mathcal{NP}$*-hard*; we therefore used the approximation algorithm in *PygMO* [8], which yields an Hypervolume value far from the exact value of a factor of 10^{-1} with probability of 0.9. For each instance, taking $\widetilde{S}^*(Y)$ as reference, we also retrieved the average *IGD+* (i.e., the average over *IGD+* computed for each of the 100 executions).

NSGA-III exhibits a good quality. Figure 4(a) shows our results based on the Hypervolume in the worst case scenario with 12+12K attributes. As expected, the quality worsens as the number of properties increases, that is, as the problem becomes more difficult. We report detailed values in Table 3 in Appendix A. According to Hypervolume, the algorithm captured ≈73% of the quality of $\widetilde{HV}^*$ on average. Figure 4(b) shows our results based on *IGD+* for the same instances used in the Hypervolume evaluation. Similar to the Hypervolume, *IGD+* becomes larger as the problem becomes more difficult, never exceeding ≈0.16. According to *IGD+*, the quality of

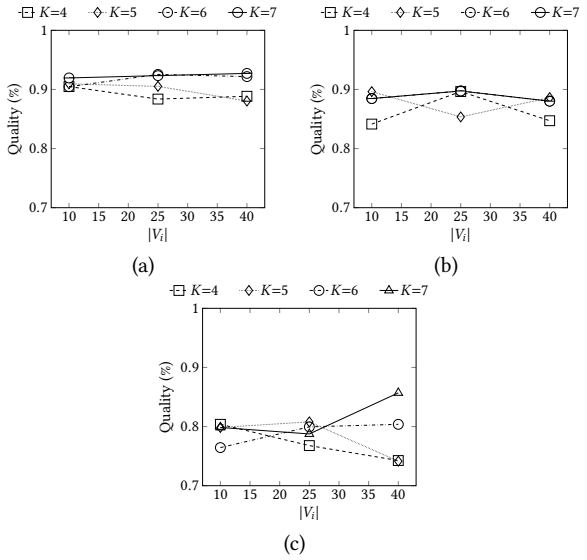

Figure 5: Quality of the best solution according to maximum strength (a), minimum cost (b), minimum penalty (c).

the algorithm is ≈0.13 on average, corresponding to an average distance between solutions in $\widetilde{S}(Y)$ and $\widetilde{S}^*(Y)$ of 0.13.

In general, Table 3 shows that Hypervolume and *IGD+* are mostly affected by the number of properties, that is, the number of objectives of Max-Property. For instance, Hypervolume has a quality of ≈85% with K=4, degrading to ≈63% with K=7, while *IGD+* value degrades from ≈0.09 to ≈0.16. We note that the decrease in quality is larger for Hypervolume since it also evaluates the *cardinality*. The cardinality of $\widetilde{S}^*(Y)$ is in fact 33× on average larger than the one of single-execution $\widetilde{S}(Y)$. More in details, $\widetilde{S}^*(Y)$ is ≈12× larger with K=4 and ≈58× with K=7.

6.2.2 Quality of the Best Solution. We evaluated the quality of the best solution returned by the three metrics in Section 4.2 with respect to the best solution returned among all executions, as follows. We denote m as the *value* of the best solution according to a specific metric of an individual execution, and m^* the best value among all executions. We averaged the quality of the three metrics over the 100 executions of each setting.

- *Maximum strength:* quality is defined as m/m^*. Maximum strength exhibits a quality of ≈91% on average, increasing to ≈92% with K=6 and K=7.
- *Minimum cost:* quality is defined as m^*/m. Minimum cost exhibits a quality of ≈86% on average, increasing to ≈87% with K=7.
- *Minimum penalty:* quality is defined as m^*/m. Minimum penalty exhibits a stable quality of ≈79%, slightly increasing to ≈79.58% with K=7.

Figure 5 shows the result for the three metrics in the worst case scenario with 12+12K attributes, while Table 4 in Appendix A reports detailed values. All metrics exhibit a stable to slightly increasing quality when varying the number of properties, despite the increasing complexity of Max-Property.

7 CONCLUSIONS

Today, the coordination between development and certification processes is more an art than a science and builds on the skills and competences of the development team, potentially impacting on the cost, time, and quality of certification. This problem is amplified by the smart service revolution that promises unmatched efficiency and effectiveness in most of our activities, at the cost of an ever-increasing reliance on digital technologies, which are now ubiquitous. In this paper, we posed the basis for a certification-driven development process that bridges the gap between software development and certification processes where *i)* certification requirements are clear from the beginning and drive the development of the system supporting them, *ii)* evidence collection points are implemented during the development process. We formulated our problem as an optimization problem and defined an heuristic approach based on a genetic algorithm solving it. Our approach retrieves a set of non-functional properties driving the implementation of certification-ready systems with highest system strength and according to an estimated budget.

ACKNOWLEDGMENTS

Research supported, in parts, by EC H2020 Project CONCORDIA GA 830927, and Università degli Studi di Milano under the program "Piano sostegno alla ricerca".

REFERENCES

[1] Marco Anisetti, Claudio A. Ardagna, Nicola Bena, and Andrea Foppiani. 2021. An Assurance-Based Risk Management Framework for Distributed Systems. In *Proc. of IEEE ICWS 2021*. Chicago, IL, USA.
[2] Marco Anisetti, Claudio A. Ardagna, and Ernesto Damiani. 2012. A Low-Cost Security Certification Scheme for Evolving Services. In *Proc. of IEEE ICWS 2012*. Honolulu, HI, USA.
[3] Marco Anisetti, Claudio Agostino Ardagna, Ernesto Damiani, and Filippo Gaudenzi. 2020. A Semi-Automatic and Trustworthy Scheme for Continuous Cloud Service Certification. *IEEE Transactions on Services Computing (TSC)* 13, 1 (2020).
[4] Marco Anisetti, Claudio A Ardagna, Filippo Gaudenzi, and Ernesto Damiani. 2019. A Continuous Certification Methodology for DevOps. In *Proc. of MEDES 2019*. Limassol, Cyprus.
[5] C.A. Ardagna, R. Asal, E. Damiani, and Q.H. Vu. 2015. From Security to Assurance in the Cloud: A Survey. *ACM Computing Surveys (CSUR)* 48, 1 (August 2015).
[6] Claudio A. Ardagna, Rasool Asal, Ernesto Damiani, Theo Dimitrakos, Nabil El Ioini, and Claus Pahl. 2021. Certification-Based Cloud Adaptation. *IEEE Transactions on Services Computing* 14, 1 (2021).
[7] Mudassar Aslam, Bushra Mohsin, Abdul Nasir, and Shahid Raza. 2020. FoNAC - An automated Fog Node Audit and Certification scheme. *Computers & Security* 93 (June 2020).
[8] Francesco Biscani and Dario Izzo. 2020. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software* 5, 53 (2020).
[9] J. Blank and K. Deb. 2020. pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020).
[10] Julian Blank, Kalyanmoy Deb, Yashesh Dhebar, Sunith Bandaru, and Haitham Seada. 2021. Generating Well-Spaced Points on a Unit Simplex for Evolutionary Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 25, 1 (2021).
[11] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. 1995. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering* 1, 1 (Dec 1995).
[12] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2018. Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases. *IEEE Transactions on Software Engineering* 44, 11 (2018).
[13] Yihai Chen, Mark Lawford, Hao Wang, and Alan Wassyng. 2013. Insulin Pump Software Certification. In *Proc. of FHIES 2013*. Macau, China.
[14] Common Criteria. [n. d.]. Certified Products. https://www.commoncriteriaportal.org/products/
[15] Kalyanmoy Deb and Ram Bhushan Agrawal. 1995. Simulated Binary Crossover for Continuous Search Space. *Complex systems* 9, 2 (1995).

[16] Kalyanmoy Deb and Samir Agrawal. 1999. A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms. In *Proc. of ICANNGA 1999*. Portorož, Slovenia.

[17] Kalyanmoy Deb and Himanshu Jain. 2014. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014).

[18] Matthias Ehrgott. 2005. *Multicriteria Optimization*. Springer-Verlag, Berlin, Germany.

[19] ENISA. 2020. Cybersecurity Certification: Candidate EUCC Scheme V1.1.1. https://www.enisa.europa.eu/publications/cybersecurity-certification-eucc-candidate-scheme-v1-1.1

[20] ENISA. 2020. EUCS – Cloud Services Scheme. https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme

[21] ENISA. 2021. Securing EU's Vision on 5G: Cybersecurity Certification. https://www.enisa.europa.eu/news/enisa-news/securing_eu_vision_on_5g_cybersecurity_certification

[22] Gabriella Gigante and Domenico Pascarella. 2012. Formal Methods in Avionic Software Certification: The DO-178C Perspective. In *Proc. of ISoLA 2012*. Heraklion, Greece.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020).

[24] Richard Hawkins, Ibrahim Habli, Tim Kelly, and John McDermid. 2013. Assurance cases and prescriptive software safety certification: A comparative study. *Safety Science* 59 (2013).

[25] Debra S Herrmann. 2002. *Using the Common Criteria for IT security evaluation*. CRC Press.

[26] IEC. 2006. *Medical device software – Software life cycle processes*. Standard. International Electrotechnical Commission, Geneva, CH.

[27] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. 2014. Difficulties in specifying reference points to calculate the inverted generational distance for many-objective optimization problems. In *Proc. of IEEE MCDM 2014*. Orlando, FL, USA.

[28] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. 2015. Modified Distance Calculation in Generational Distance and Inverted Generational Distance. In *Proc. of EMO 2015*. Guimarães, Portugal.

[29] Sharmin Jahan, Ian Riley, Charles Walter, Rose F. Gamble, Matt Pasco, Philip K. McKinley, and Betty H.C. Cheng. 2020. MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases. *Future Generation Computer Systems* 109 (2020).

[30] Himanshu Jain and Kalyanmoy Deb. 2014. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014).

[31] Samuel Paul Kaluvuri, Michele Bezzi, and Yves Roudier. 2013. Bringing Common Criteria Certification to Web Services. In *Proc. of IEEE SERVICES 2013*. Santa Clara, CA, USA.

[32] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems*. Springer.

[33] Miqing Li and Xin Yao. 2019. Quality Evaluation of Solution Sets in Multiobjective Optimisation: A Survey. *ACM Computing Surveys (CSUR)* 52, 2 (2019).

[34] Sebastian Lins, Stephan Schneider, Jakub Szefer, Shafeeq Ibraheem, and Ali Ali. 2019. Designing Monitoring Systems for Continuous Certification of Cloud Services: Deriving Meta-requirements and Design Guidelines. *Comm. of the Association for Information Systems* 44 (2019).

[35] Sara N. Matheu, José L. Hernández-Ramos, Antonio F. Skarmeta, and Gianmarco Baldini. 2020. A Survey of Cybersecurity Certification for the Internet of Things. *ACM Computing Surveys (CSUR)* 53, 6 (Dec. 2020).

[36] Sara N. Matheu-García, José L. Hernández-Ramos, Antonio F. Skarmeta, and Gianmarco Baldini. 2019. Risk-based automated assessment and testing for the cybersecurity certification and labelling of IoT devices. *Computer Standards & Interfaces* 62 (February 2019).

[37] Dominique Méry and Neeraj Kumar Singh. 2010. Trustable Formal Specification for Software Certification. In *Proc. of ISoLA 2010*. Heraklion, Greece.

[38] Kaisa Miettinen. 2008. Introduction to Multiobjective Optimization: Noninteractive Approaches. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer Berlin Heidelberg.

[39] Håvard Myrbakken and Ricardo Colomo-Palacios. 2017. DevSecOps: A Multivocal Literature Review. In *Proc. of SPICE 2017*. Palma de Mallorca, Spain.

[40] Daniel Port and Joel Wilf. 2013. The Value of Certifying Software Release Readiness: An Exploratory Study of Certification for a Critical System at JPL. In *Proc. of ACM/IEEE ESE, 2013*. Baltimore, MD, USA.

[41] Leanna Rierson. 2017. *Developing safety-critical software*. CRC Press.

[42] F. Ruiz, M. Luque, and J. M. Cabello. 2009. A classification of the weighting schemes in reference point procedures for multiobjective programming. *Journal of the Operational Research Society* 60, 4 (2009).

[43] Philipp Stephanow, Gaurav Srivastava, and Julian Schütte. 2016. Test-Based Cloud Service Certification of Opportunistic Providers. In *Proc. of IEEE CLOUD 2016*. San Francisco, CA, USA.

[44] Christoph Torens, Florian-M. Adolf, and Lukas Goormann. 2014. Certification and Software Verification Considerations for Autonomous Unmanned Aircraft. *Journal of Aerospace Information Systems* 11, 10 (2014).

[45] Elaine Venson, Xiaomeng Guo, Zidi Yan, and Barry Boehm. 2019. Costing Secure Software Development: A Systematic Mapping Study. In *Proc. of ARES 2019*. Canterbury, UK.

A ADDITIONAL EXPERIMENTAL RESULTS

Table 3 shows detailed results of Hypervolume and *IGD+* experiments. The two indicators degrade as the number of objectives increase, and they are virtually unaffected by variations in the number of attributes and attributes values.

Table 3: Hypervolume and *IGD+* for all settings

K	N. Attr	$\|V_i\|$	$\widetilde{\text{HV}}$qual.(%)	*IGD+*	K	N. Attr	$\|V_i\|$	$\widetilde{\text{HV}}$qual.(%)	*IGD+*
4	4+4K	10	78.7	0.11	6	4+4K	10	33.2351	0.1616
4	4+4K	25	80.9	0.1	6	4+4K	25	36.6431	0.1674
4	4+4K	40	81.19	0.11	6	4+4K	40	26.8122	0.1432
4	8+8K	10	90.89	0.07	6	8+8K	10	32.3279	0.1402
4	8+8K	25	80.43	0.1	6	8+8K	25	21.1256	0.1166
4	8+8K	40	94.43	0.05	6	8+8K	40	29.0331	0.137
4	12+12K	10	80.48	0.1	6	12+12K	10	28.3816	0.1362
4	12+12K	25	87.42	0.08	6	12+12K	25	29.4604	0.1294
4	12+12K	40	91.56	0.06	6	12+12K	40	27.2143	0.1338
AVG			85.11	0.09	*AVG*			70.76	0.14
5	4+4K	10	76.1	0.12	7	4+4K	10	57.92	0.17
5	4+4K	25	72.87	0.13	7	4+4K	25	62.06	0.16
5	4+4K	40	68.07	0.15	7	4+4K	40	66.62	0.16
5	8+8K	10	69.33	0.15	7	8+8K	10	62.44	0.16
5	8+8K	25	69.43	0.15	7	8+8K	25	63.82	0.18
5	8+8K	40	72.09	0.13	7	8+8K	40	62.53	0.16
5	12+12K	10	70.41	0.14	7	12+12K	10	63.99	0.16
5	12+12K	25	65.03	0.16	7	12+12K	25	67.91	0.16
5	12+12K	40	82.36	0.11	7	12+12K	40	63.5	0.15
AVG			71.74	0.14	*AVG*			63.42	0.16

	$\widetilde{\text{HV}}$qual.(%)	*IGD+*
AVG	72.76	0.13

Table 4 shows detailed results of quality evaluation based on the three selection metrics in Section 4.2. They all show a consistent quality across all settings.

Table 4: Solution selection quality for all settings

K	N. Attr	$\|V_i\|$	Str.(%)	Cost(%)	Pen.(%)	K	N. Attr	$\|V_i\|$	Str.(%)	Cost(%)	Pen.(%)
4	4+4K	10	90.89	87.05	81.77	6	4+4K	10	92.42	88.39	84.73
4	4+4K	25	90.88	80.06	79.5	6	4+4K	25	92.55	80.55	82.15
4	4+4K	40	92.48	83.58	72.62	6	4+4K	40	93.04	78.12	81.95
4	8+8K	10	91.61	82.27	84.34	6	8+8K	10	91.89	89.35	78.63
4	8+8K	25	90.7	85.02	81.88	6	8+8K	25	88.66	88.9	76.05
4	8+8K	40	88.98	79.83	78.25	6	8+8K	40	87.44	82.42	70.66
4	12+12K	10	90.49	84.14	80.38	6	12+12K	10	90.46	93.67	76.45
4	12+12K	25	88.37	89.65	76.79	6	12+12K	25	92.54	88.3	79.95
4	12+12K	40	88.84	84.71	74.26	6	12+12K	40	92.17	89.68	80.38
AVG			90.36	84.03	78.86	*AVG*			91.24	86.6	78.99
5	4+4K	10	91.68	86.36	75.33	7	4+4K	10	93.19	86.84	81.29
5	4+4K	25	90.52	83.15	82.92	7	4+4K	25	91.52	80.09	78.06
5	4+4K	40	90.82	83.83	77.54	7	4+4K	40	91.57	82.88	83.97
5	8+8K	10	89.12	89.6	77.86	7	8+8K	10	89.41	91.23	68.16
5	8+8K	25	90.98	89.73	79.8	7	8+8K	25	90.1	90.42	80.36
5	8+8K	40	89.57	82.66	78.97	7	8+8K	40	92.26	88.48	80.08
5	12+12K	10	90.95	89.63	79.85	7	12+12K	10	91.95	88.46	79.85
5	12+12K	25	90.52	85.36	80.8	7	12+12K	25	92.34	89.74	78.75
5	12+12K	40	88.04	88.61	74.14	7	12+12K	40	92.71	88.02	85.7
AVG			90.24	86.55	78.58	*AVG*			91.67	87.35	79.58

	Str.(%)	Cost(%)	Pen.(%)
AVG	90.88	86.13	79