BACRank: Ranking Building Automation and Control System Components by Business Continuity Impact

Herson Esquivel-Vargas[1](✉), Marco Caselli[2], Erik Tews[1], Doina Bucur[1], and Andreas Peter[1]

[1] University of Twente, Enschede, The Netherlands
{h.esquivelvargas,e.tews,d.bucur,a.peter}@utwente.nl
[2] Siemens AG, Munich, Germany
marco.caselli@siemens.com

Abstract. Organizations increasingly depend on Building Automation and Control Systems (BACSs) to support their daily tasks and to comply with laws and regulations. However, BACSs are prone to disruptions caused by failures or active attacks. Given the role BACSs play in critical locations such as airports and hospitals, a comprehensive impact assessment methodology is required that estimates the effect of unavailable components in the system. In this paper, we present the foundations of the first impact assessment methodology for BACSs focused on business continuity. At the core of our methodology, we introduce a novel graph centrality measure called BACRank. We quantify the contribution of BACS components to different business activities. Moreover, we take functional dependencies among components into account to estimate indirect consequences throughout the infrastructure. We show the practical applicability of our approach on a real BACS deployed at a 5-story building hosting 375 employees on an international university campus. The experimental evaluation confirms that the proposed methodology successfully prioritizes the most relevant components of the system with respect to the business continuity perspective.

1 Introduction

Operational Technology (OT), and specifically Building Automation and Control Systems (BACSs), are steadily increasing in number and complexity [18]. Many organizations depend on BACSs to comply with laws and regulations required to operate [3–5]. Thus, the dependability of BACSs is crucial for their daily operation. However, complex systems with extended uptimes are prone to occasional outages due to failures or active attacks.

Unavailable BACS components have direct consequences on the services they are part of, and indirect consequences that can spread throughout neighboring components that rely on them to execute their functions. Knowing the impact of unavailable BACS components help organizations to better prepare and react upon those undesired events. From the *preventive* perspective, they can compute incident probabilities and obtain risk estimations (Risk = Impact × Probability). Risks are then used as an input to establish contingency plans and to decide on improvement strategies. On the other hand, the increasing interest in monitoring tools for BACSs might lead to an overwhelming number of alerts that must be managed by building administrators [11, 12]. In this regard, from the *reactive* perspective, the impact measurement can be used to prioritize failure and security alerts, which helps to efficiently allocate resources to solve the problems.

Measuring the impact of unavailable IT components is a well understood problem [13]. It is not clear, however, how to implement impact assessments for OT systems like BACSs, that extend beyond the *cyber* domain to the *physical* world. From the security perspective, the situation degenerates considering that most of the reported attacks on OT systems target their availability [1]. Although in principle, BACSs could be assessed using business continuity methodologies, the peculiarities of BACSs must be taken into consideration to develop a comprehensive impact assessment in this domain.

1.1 Related Work

Impact assessments focus on diverse target goals ranging from environmental to economical impact, physical damage, and business continuity, just to mention a few examples [6, 8, 13, 17, 20]. ISO 27031 describes a consolidated methodology to perform impact assessments in the IT domain focused on business continuity [13]. The analyzed assets are typically devices such as PCs, switches, and servers. Those assets are linked to one or more pre-scored business activities in order to assign them the highest score among the related activities. The outcome is a ranking of IT assets prioritized with respect to the relevance of the business activities they participate in.

OT, on the other hand, lacks the maturity level of standardized methodologies for impact assessments. To fill this gap, a number of academic works have been proposed, mostly for Industrial Control Systems (ICSs) [6, 8, 20]. These works are based on the observation of cause-and-effect relations between data points representing sensors and actuators. The observations are taken from simulated environments where changes are induced to log the corresponding effects. Knowing the limits of the physical process (e.g., what is the threshold before the power plant explodes?), data points with the higher potential to reach those limits are prioritized.

The only work that is focused on BACSs prioritizes component categories rather than individual assets [23]. They automatically analyze work orders describing building's routine and maintenance operations. Based on the information recorded in the work orders, such as location, problem description, and priority, they rank equipment categories like "fans", "valves", "pumps", etc.

The approach presented in this paper aims to adapt previous works to the context of BACSs and to solve practical limitations that hinder their implementation in real BACSs. Our impact assessment is focused on *business continuity* as it is commonly done for IT systems. Nonetheless, our assets are not physical devices since we consider this approach too coarse grained. Neither are individual data points, as in the ICS prototypes, since such fine grained analysis might suffer from scalability problems in real-life systems. We chose *software modules* as a middle ground generic abstraction that provides a suitable granularity level. Instead of measuring the propagation effect via cause-and-effect experiments, unlikely to be allowed in real BACS deployments, we use *functional dependencies* among the software modules. Finally, although many software modules could be clustered by functional similarity, we acknowledge that the role they play for different business activities makes a crucial difference. Thus, we reference and prioritize software modules individually.

1.2 Contribution

We present the first impact assessment methodology for BACSs focused on business continuity. We adapt and integrate standard business continuity methodologies from the IT domain, and combine them with software analysis techniques. From the IT domain, we follow the standard procedures used to score business activities. After mapping software modules with business activities, we use the activities' score to derive the related modules' score. From the software engineering domain, we implement a module dependency analysis that aims to estimate the propagation effect of unavailable modules.

Our impact assessment methodology models BACS software modules as vertices in a graph data structure where the edges represent functional dependencies among modules. A quantitative measurement of a node's relevance in a graph is called a node's *centrality* and it is computed by means of graph centrality measures [25]. Our impact assessment methodology formally defines the requirements such centrality measure must satisfy in order to quantify the software modules' impact. Additionally, we implement an instance of such centrality measure and we call it *BACRank*. BACRank scores software modules in a dependency graph according to their relevance from the business continuity perspective. Our notion of "relevant" includes those software modules that are: 1) needed by core business processes; and 2) needed by other relevant modules.

Finally, we evaluate BACRank in a real-world BACS deployed at a 5-story office building hosting 375 employees on an international university campus. The underlying BACS graph is comprised of 160 software modules and 412 module dependencies. Such evaluation confirms that BACRank prioritizes relevant software modules according to the defined relevance notion.

2 Building Automation and Control Systems

Modern buildings provide more than a physical space to their occupants. Environmental conditions are controlled by heating, cooling, and ventilation systems. Indoor transportation of goods and people is done through escalators, elevators, and travelators. Other services like CCTV, alarms, and physical access control are also common. Control engineers implement those services and many more in *Building Automation and Control Systems* (BACSs). BACSs offer functionality that brings comfort and convenience to the users while unified control and energy efficiency engage building managers.

In BACSs, IT networking infrastructure is used to interconnect all the subsystems in a building [19]. We distinguish three levels in the typical BACS architecture as shown in Fig. 1. The *management* level provides monitoring and control functions to building administrators. The *automation* level is comprised of embedded systems called BACS controllers that implement the logic behind the building services. BACS controllers receive inputs from the environment through sensors, execute the appropriate logic, and send outputs back to the environment using actuators. Both sensors and actuators are the elements found in the *field* level. Dashed lines around field level components denote that software modules at the automation level are linking them. Building services are comprised of one or more — possibly interacting — software modules.

Organizations typically have specific needs from their BACS. Deciding which building services are to be implemented depends on the organization's purpose. For example,

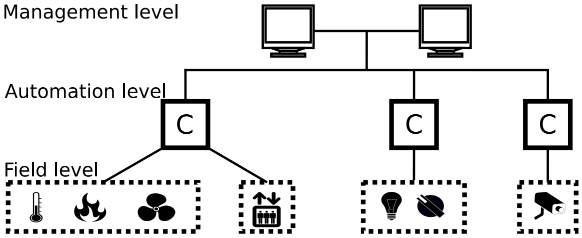


Fig. 1. Three-layer architecture of IT networks supporting BACSs.

the building services required in a hospital are quite different from the services required in an office building. Thus, let us start with a brief discussion on the role that BACSs play in organizations.

Organizations Requirements. All organizations have a goal to pursue. Non-trivial goals require a divide-and-conquer approach that splits the task at hand into smaller activities. The attainment of the organization's goal depends on the success of the individual activities. In what follows, we refer to those activities as *business processes*.

Business processes make use of diverse resources like people, supplies, and assets. Buildings are one of those resources, and fostering business processes became the main task of BACSs. BACSs, just as other technological projects like software systems, are the answer to specific business process needs. While some of those needs are nice-to-have features (perhaps comfort oriented), others constitute necessities that must be fulfilled [3–5]. In what follows we will refer to the former type of building services as *supporting services* and to the latter as *enabling services*.

Building Services Dependability. Building services, and particularly enabling services, must be dependable. Dependability involves, among others, concepts such as privacy of the data, message integrity, and in general the availability of the system, meaning that it behaves as it was designed and provides timely responses [15, 21].

Previous research has identified *availability* as the most important feature in diverse OT systems such as smart-grids and Industrial Control Systems (ICS) [9, 16, 24]. This conclusion seems to match the trend of real-life attacks on OT systems, typically oriented towards Denial-of-Service [1]. ISO standard 27031 also recognizes the role that OT systems play in business continuity by adding an annex on "High availability embedded systems" [13]. Although an impact assessment could be tailored to any of the aforementioned dependability factors, we focus on the availability of building services, their underlying software modules, and the supported business processes.

3 System Model and Information Requirements

We consider mature organizations and critical buildings whose dependence on BACSs is crucial for their daily operations. Such organizations have undergone — or are willing to apply — business continuity methodologies like those proposed in ISO standards 22301 and 27031. Although an ISO certification is not required, critical buildings such as hospitals and airports are typically demanded to implement business continuity methodologies like those described in the standards [10, 22].

On the technical side, our approach is independent of the BACS protocol in use and the underlying communication methods. We consider standard BACSs comprised of sensors and actuators connected to BACS controllers, which contain the logic behind the smart building, as in Fig. 1.

To implement a comprehensive BACS impact assessment focused on business continuity, technical and business aspects must be taken into consideration. First, it is required to make explicit the relation between BACS services and business activities. We propose to do so by considering their physical overlap in the building. Two views of the building layout are needed: one segregated by business processes and the other segregated by the area of influence of the building services. We put the two views of the building layout on top of each other to unveil the mapping. Moreover, we need to quantify the support of building services on the related business processes. The quantification can be expressed as a percentage where 100% means that the service in question is an *enabling service*, whereas lower values characterize *supporting services* (see Sec. 2).

The second aspect is to know which business activities are critical for the organization, so we can map such criticality status to the corresponding BACS services. Business activity priorities can be found in business continuity plans, where the activity scores are calculated by means of a procedure called Business Impact Analysis (BIA). The main purpose of the BIA is to score each business process based on questionnaires answered by process managers. The idea is to estimate the impact of a business process halted during different time ranges (e.g., 0–2 hours, 2–8 hours, etc.), considering diverse impacts like customer service, legal, financial, and reputation. The managers assign a severity level to each impact of the processes they are in charge of. Finally, the average level score is computed for all processes to determine their relevance with respect to the organization's goal. For further details on the BIA, we refer the reader to the ISO standards 22301 and 27031 [13, 14]. In summary, the BIA lists all business processes, their relevance scores according to the business perspective, and a calendar that specifies their execution period.

Finally, the third aspect lies on the technical side. It is important to know *when* the building services are actually needed since they might not be relevant out of their duty periods (e.g., the heating service during summer). Furthermore, it is crucial to understand *how* the building services are implemented to find possible design flaws that could lead to availability issues (e.g., single points of failure). To obtain this information, we assume access to the BACS design documentation to identify 1) all building services, 2) their duty cycles, 3) the underlying software modules, and 4) the functional dependencies among modules. Since some dependencies might be stronger than others, a quantification is needed in this regard. We propose a percentage value as a simple mechanism to denote the dependence strength, where 100% means that the dependent module cannot operate without the other.

Table 1 summarizes the information requirements. While most of the information is typically already available in mature and critical organizations, we emphasize three components where an expert's judgment is required: 1) the building services' support on business processes; 2) the software modules' dependency strength; and 3) the building services calendar. It is worth noting that, if needed, an individual module's calendar can be derived from the calendar of the building service it belongs to.

The parameters described in this section provide the basic input that allows us to account for the impact, in terms of availability, of business processes supported by building services comprised of software modules. Those parameters are not only highly meaningful for the purpose of an impact assessment, but are typically already present in critical organizations. Thus, reducing the effort of implementing the methodology proposed in this paper. We do not discard, however, that other business or technical aspects could be included to complement or replace some elements in the proposed list, while the core principles of our methodology prevail.

Table 1. Information requirements summary. Expert-based information shown in *italics*.

Business Information	BACS Technical Information
Business continuity plan	Engineering design
↳ BIA	↳ Building services list
↳ Business process list	↳ *Calendar*
↳ Score	↳ Software modules
↳ Calendar	↳ Dependencies
	↳ *Strength*
Building layout	Building layout
↳ Segregated by business processes	↳ Segregated by building services
↳ *Building services support on the overlapping business processes* ↲	

4 Impact Assessment Methodology

We abstract the BACS as a directed graph data structure where software modules are represented by vertices and their functional dependencies are the edges. The edge direction denotes the way information flows and its weight represents the dependency strength. Formally, the BACS is defined as a graph $G(V, E)$ where V is a nonempty set of vertices (or nodes) and E is a set of edges. Each edge has exactly two vertices in V as endpoints since self-dependencies (i.e., loops) are implicit for all modules. An edge $e \in E$ is represented as $e_{u,v}$ where u and v denote the origin and the destination of the edge, respectively. Edge weights are represented as a function $\omega : E \rightarrow [0, 1]$ that assigns each edge $e \in E$ a weight $\omega(e)$. The set of edges with destination $m \in V$ is defined as $\Gamma^{-}(m)$ and the set of vertex origins in $\Gamma^{-}(m)$ is $N^{-}(m)$. Analogously, the set of edges with origin m is defined as $\Gamma^{+}(m)$ and the set of vertex destinations in $\Gamma^{+}(m)$ is $N^{+}(m)$.

We aim to measure the impact of BACS software modules based on their relevance to the availability of business processes and their functional dependencies with other modules. The identification of important vertices in a graph is done by means of graph centrality metrics. Therefore, our impact assessment methodology can be modeled as a graph centrality measure.

We propose a graph centrality measure comprised of two parts. First, a set up procedure that assigns vertices an initial score based on the BIA score of the related business processes and the module's support to those business processes. Second, a graph centrality measure that contemplates the propagation effect of unavailable modules. Module's rank positions are based on their final impact scores. The next sections detail both parts.

4.1 Initial Score

Notation. The set of business processes running in the building is defined as $P = \{p_1, ..., p_n\}$. The set of building services offered is defined as $S = \{s_1, ..., s_m\}$. The BIA score assigned to each business process is defined as a function $\beta : P \to [0, 1]$, where $\beta(p_i)$ for any $p_i \in P$ is proportional to p_i's relevance for the organization. Given an arbitrary $s_i \in S$ and $p_j \in P$, the estimated support s_i provides to p_j is defined as a function $\gamma : S \times P \to [0, 1]$, where higher values denote stronger support. Since building services are comprised of one or more software modules, we can formally state that $s_j \subseteq V$ for any $s_j \in S$. For an arbitrary module $m \in V$ that is part of service $s_j \in S$, the support m provides to an arbitrary $p_k \in P$ is given by $\gamma(s_j, p_k)$. Finally, we define a *time* function that takes two inputs: 1) the object whose calendar is going to be inspected (either a business process or a software module); and 2) the current time t. The output is binary and indicates whether the object given as first input is running/needed or not at time t, denoted by a 1 or a 0, respectively.

Initial Score. The initial score given to each software module in the graph, labels vertices with a numerical value that summarizes three important aspects: 1) the relevance of the related business processes represented by function β; 2) the module's support to each business process represented by function γ; and 3) the time in which both, the business process is running *and* the software module is needed (according to its building service calendar).

To determine the influence that module $m \in V$ has over each business process, we multiply $\beta(p_i) \cdot \gamma(s_j, p_i)$ for all $p_i \in P$, given that m is part of $s_j \in S$. Computing the initial score for module m at time t, denoted $\delta(m, t)$, consists of taking the maximum influence found among active business processes, given that module m is also active at time t. Formally,

$$\delta(m,t) = \begin{cases} \max_{1 \le i \le n}(\beta(p_i) \cdot \gamma(s_j, p_i)) & \text{if time}(p_i, t) = \text{time}(m, t) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

4.2 Graph Centrality Measure

We propose to estimate the propagation effect of unavailable modules by means of a graph centrality measure. Before describing the requirements for such centrality measure we introduce some definitions.

Definition (Module *equivalence*). Two modules $m_1, m_2 \in V$ are *equivalent (at time t)* (denoted as $m_1 \equiv m_2$) if all of the following properties hold:

$$N^+(m_1) = N^+(m_2) \tag{1}$$

$$N^-(m_1) = N^-(m_2) \tag{2}$$

$$\forall e_{m_1,n} \in \Gamma^+(m_1), e_{m_2,n} \in \Gamma^+(m_2) : \omega(e_{m_1,n}) = \omega(e_{m_2,n}) \tag{3}$$

$$\forall e_{n,m_1} \in \Gamma^-(m_1), e_{n,m_2} \in \Gamma^-(m_2) : \omega(e_{n,m_1}) = \omega(e_{n,m_2}) \tag{4}$$

$$\delta(m_1, t) = \delta(m_2, t) \tag{5}$$

Definition (Module *equivalence with exception*). Two modules $m_1, m_2 \in V$ are called *equivalent with exception* if at least one of the above equivalence properties is violated. In this case, we explicitly mention the exception and denote this as $m_1 \equiv_e m_2$ (exception).

In what follows we define three basic requirements that a centrality measure $\Delta(m, t)$ for module m at time t, must satisfy to quantify the impact of BACS software modules.

1. For any two active modules one of which, ceteris paribus, has higher initial score, must score higher.
The aim of the first requirement is to ensure that the impact score difference of two modules with identical topological features in the graph is determined by their initial score. Formally described in eq. 6, for all active modules m_1, m_2:

$$m_1 \equiv_e m_2 \ (\delta(m_1, t) > \delta(m_2, t)) \Rightarrow \Delta(m_1, t) > \Delta(m_2, t) \tag{6}$$

2. For any two active modules one of which, ceteris paribus, sends information to an active module with higher impact score than its counterpart, must score higher.
The goal of the second requirement is to acknowledge that feedback plays an important role in software module dependency graphs. Unlike web centrality measures, where the feedback contribution comes from a node's incoming edges [7], software modules in our setting get their contribution from the modules they send information to. The rationale being that the receiving modules depend on that input to execute their functions. Formally described in eq. 7, for all active modules m_1, m_2:

$$\begin{aligned} m_1 \equiv_e m_2 \ (\exists n_1 \in N^+(m_1), n_2 \in N^+(m_2) : N^+(m_1) \setminus \{n_1\} = N^+(m_2) \setminus \{n_2\} \wedge \\ \omega(e_{m_1,n_1}) = \omega(e_{m_2,n_2}) \wedge \Delta(n_1, t) > \Delta(n_2, t)) \Rightarrow \Delta(m_1, t) > \Delta(m_2, t) \end{aligned} \tag{7}$$

3. For any two active modules one of which, ceteris paribus, sends information to an active module with stronger dependency than its counterpart, must score higher.
The purpose of the third requirement is to emphasize that the link strength regulates the fraction of the impact score to be transferred from the destination vertex to the source vertex. Formally described in eq. 8, for all active modules m_1, m_2:

$$m_1 \equiv_e m_2 \ (\exists! n' \in N^+(m_1) = N^+(m_2) : \omega(e_{m_1,n'}) > \omega(e_{m_2,n'})) \Rightarrow \Delta(m_1, t) > \Delta(m_2, t) \tag{8}$$

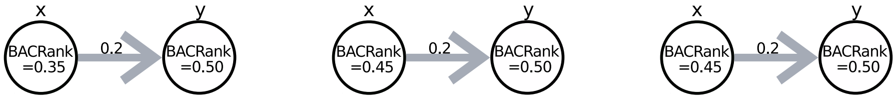
4.3 BACRank

Taking into account the requirements stated before, we define a new graph centrality measure called BACRank. The BACRank score of vertex m at time t, is computed as its initial score $\delta(m, t)$ plus a contribution from the vertices m points to. From those vertices, m will get a percentage of their BACRank score determined by the strength of the link, represented by $\omega(e_{m,n})$. This mechanism boosts the scores of vertices that are highly important from the technical and business process availability standpoint. The algorithm is defined as

$$\text{BACRank}(m, t; i) = \begin{cases} \delta(m, t), \text{ at iteration } i = 0, \\ \delta(m, t) + \sum_{n \in N^+(m)} \text{BACRank}(n, t; i-1) \cdot \omega(e_{m,n}), \text{ for } i > 0. \end{cases}$$

To keep BACRank scores bounded, all scores are normalized in the range [0, 1] after every iteration. The algorithm is said to converge if for all vertices the score difference in two consecutive iterations is less than a small value ε. An empirical convergence proof is provided in Sec. 5 as shown in Fig. 4. The final score, simply denoted as BACRank(m, t) by leaving out the iteration count i, is then iteratively computed until iteration i such that the difference between BACRank$(m, t; i)$ and BACRank$(m, t; i - 1)$ is smaller than ε.

Figure 2 shows a simple BACRank execution example. At iteration 0, in Fig. 2a, the BACRank score of software modules x and y is their corresponding initial score δ, in this example setting equals 0.35 and 0.50, respectively. At iteration 1, in Fig. 2b, module x has increased its BACRank score by 0.10, which is the result of taking 20% (edge weight) of module y's score at iteration 0. Module y remains with the same score since it does not have any outgoing edges. Convergence is already reached at iteration 2 (Fig. 2c) because the BACRank values did not change with respect to the previous iteration.



(a) Iteration 0. **(b)** Iteration 1. **(c)** Iteration 2.

Fig. 2. Simple execution example of the BACRank algorithm.

In what follows, we formally prove the requirements from Sec. 4.2. For the sake of brevity, in the proofs we refer to BACRank simply as BR.

Proof of requirement 1. Let m_1, m_2 be two active modules such that:

$$m_1 \equiv_e m_2 \; (\delta(m_1, t) > \delta(m_2, t)) \tag{9}$$

Then the following holds for any iteration $i > 0$:

$$\begin{aligned} \mathrm{BR}(m_1, t; i) &> \delta(m_2, t) + \sum_{n \in N^+(m_1)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_1,n}) && \text{(by (9))} \\ &= \delta(m_2, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_1,n}) && \text{(by equiv. prop. (1))} \\ &= \delta(m_2, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) = \mathrm{BR}(m_2, t; i) && \text{(by equiv. prop. (3))} \end{aligned}$$

Proof of requirement 2. Let m_1, m_2 be two active modules such that:

$$m_1 \equiv_e m_2 \; (\exists n_1 \in N^+(m_1), n_2 \in N^+(m_2) : N^+(m_1) \setminus \{n_1\} = N^+(m_2) \setminus \{n_2\} \wedge \\ \omega(e_{m_1,n_1}) = \omega(e_{m_2,n_2}) \wedge \mathrm{BR}(n_1, t; i) > \mathrm{BR}(n_2, t; i)) \tag{10}$$

Then the following holds for any iteration $i > 0$:

$$\begin{aligned}
\mathrm{BR}(m_1, t; i) &> \delta(m_1, t) + \sum_{n \in N^+(m_1) \setminus \{n_1\}} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_1,n}) + \mathrm{BR}(n_2, t; i) \cdot \omega(e_{m_2,n_2}) \quad \text{(by (10))} \\
&= \delta(m_1, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) \qquad \text{(by (10) and equiv. prop. (3))} \\
&= \delta(m_2, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) = \mathrm{BR}(m_2, t; i) \ \text{(by equiv. prop. (5))}
\end{aligned}$$

Proof of requirement 3. Let m_1, m_2 be two active modules such that:

$$m_1 \equiv_e m_2 \ (\exists! n' \in N^+(m_1) = N^+(m_2) \,:\, \omega(e_{m_1,n'}) > \omega(e_{m_2,n'})) \tag{11}$$

Then the following holds for any iteration $i > 0$:

$$\begin{aligned}
\mathrm{BR}(m_1, t; i) &> \delta(m_1, t) + \sum_{n \in N^+(m_1) \setminus \{n'\}} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_1,n}) + \mathrm{BR}(n', t; i) \cdot \omega(e_{m_2,n'}) \quad \text{(by (11))} \\
&= \delta(m_1, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) \qquad \text{(by equiv. prop. (1))} \\
&= \delta(m_2, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) = \mathrm{BR}(m_2, t; i) \ \text{(by equiv. prop. (5))}
\end{aligned}$$

5 Experimental Evaluation

Environment Description. We executed BACRank on the BACS of a 5-story office building on an international university campus, hosting about 375 employees in 252 rooms. The local building manager provided assistance with the required technical information. We identified 12 business processes that take place in this building, some of them running only at specific periods of the year. The BIA revealed the corresponding scores as presented in Table 2.

The core BACS is implemented using the BACnet protocol [2]. The BACnet system controls the heating, ventilation, cooling, and lighting services. Other building services, such as physical access control, are implemented with different protocols and tools we did not have access to. We consider here only the building services implemented in BACnet.

The BACnet system is comprised of 160 software modules running in 5 multi-purpose controllers (BACnet profile B-BC) and 28 application-specific controllers (BACnet profile B-ASC). Figure 3 shows the software modules dependency graph where 22 vertices are isolated and the remaining 138 are connected in the main subgraph. Vertex colors indicate the device they run in. Red modules are part of the heating controller, whereas blue modules are part of the cooling controller. The lighting system is controlled by the yellow modules. Green modules run in a controller in charge of multiple services (ventilation, heating, and cooling) throughout the building. Purple modules also implement multiple services but in one specific location of the building. Finally, each gray module represents one application-specific controller running exactly one software module (thermostats).

Results. We used weekly time resolution in our experiments based on the activity of the business processes and software modules analyzed. This implies that a new ranking has to be computed every week of the year to take into account business processes that start or stop execution, and software modules that might or not be needed (e.g., due to changes in climate conditions). For each week, BACRank is executed a number of iterations until the scores' convergence is reached. Figure 4 shows the quick convergence of BACRank on the real BACS graph. After the tenth iteration, on average, the difference between two consecutive scores (ϵ) is smaller than 0.0006. After 60 iterations $\epsilon = 0$. To run our evaluation we defined an $\epsilon < 10^{-6}$, which implies that 20 iterations are needed.

Table 2. Business Processes (BPs) and their corresponding BIA scores.

Nº	Business Process	Score	Nº	Business Process	Score
1	Research	.63	7	Introduction week	.60
2	Application / admission	.27	8	Administrative support	.47
3	Accounting	.60	9	Education advisory	.53
4	Technical support	.43	10	Marketing & communication	.43
5	Courses and others (periodic)	.73	11	Catering	1.0
6	Trainings and others (non-periodic)	.70	12	Student associations	.50

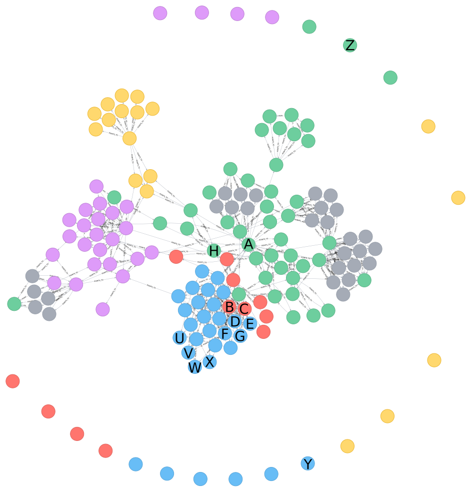


Fig. 3. Real software modules dependency graph.

Figure 5a illustrates the 52 rankings obtained throughout the year, where each colored line represents a software module (following the same color scheme used in Fig. 3). The vertical axis represents the ranking position where top ranked modules start at position 1. Modules with identical scores in Fig. 5a are randomly assigned a slot next to their analogous. Figure 5b on the other hand, shows the actual BACRank score for each module. The impression of having fewer lines in Fig. 5b than in Fig. 5a is due to multiple overlapping lines (i.e., modules with identical score).

BACRank successfully identifies software modules that are part of relevant business processes, and are required by other relevant modules in the infrastructure. Throughout the experiments, module “Multi-purpose Substation” (vertex A in Fig. 3) was considered the most important because, among others, it supports the most important business process according to the BIA (BP_{11}) *and* it provides information required by 19 other important modules in 7 different devices. Vertex A is depicted as an horizontal green line at the top of Fig. 5a and Fig. 5b.

At the bottom of the ranking there is a set of approximately 30 modules consistently low ranked. Some of them, starting from the last one — “AirExtraction” — and ascending with “Electricitymeter Experiments”, “CoolSection [sect. numbers 1–4]_Log”, are shown in Fig. 3 as Z, Y, X, W, V, and U, respectively. There are two aspects that justify their poor scoring performance. First, a vertex with out-degree of 0 is likely to be low ranked because an important source of BACRank score comes from other vertices that depend on the module in question. Second, if no other modules depend on it, its BACRank score comes exclusively from the *initial score* which is, in turn, based on the related business processes and the module’s support to them. If there are no related business processes, as in the case of safety oriented modules; or the module’s participation in the business processes is marginal, then the module in question will get a low BACRank score.

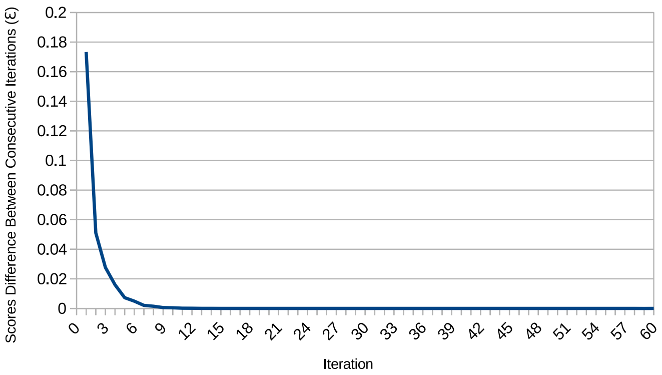
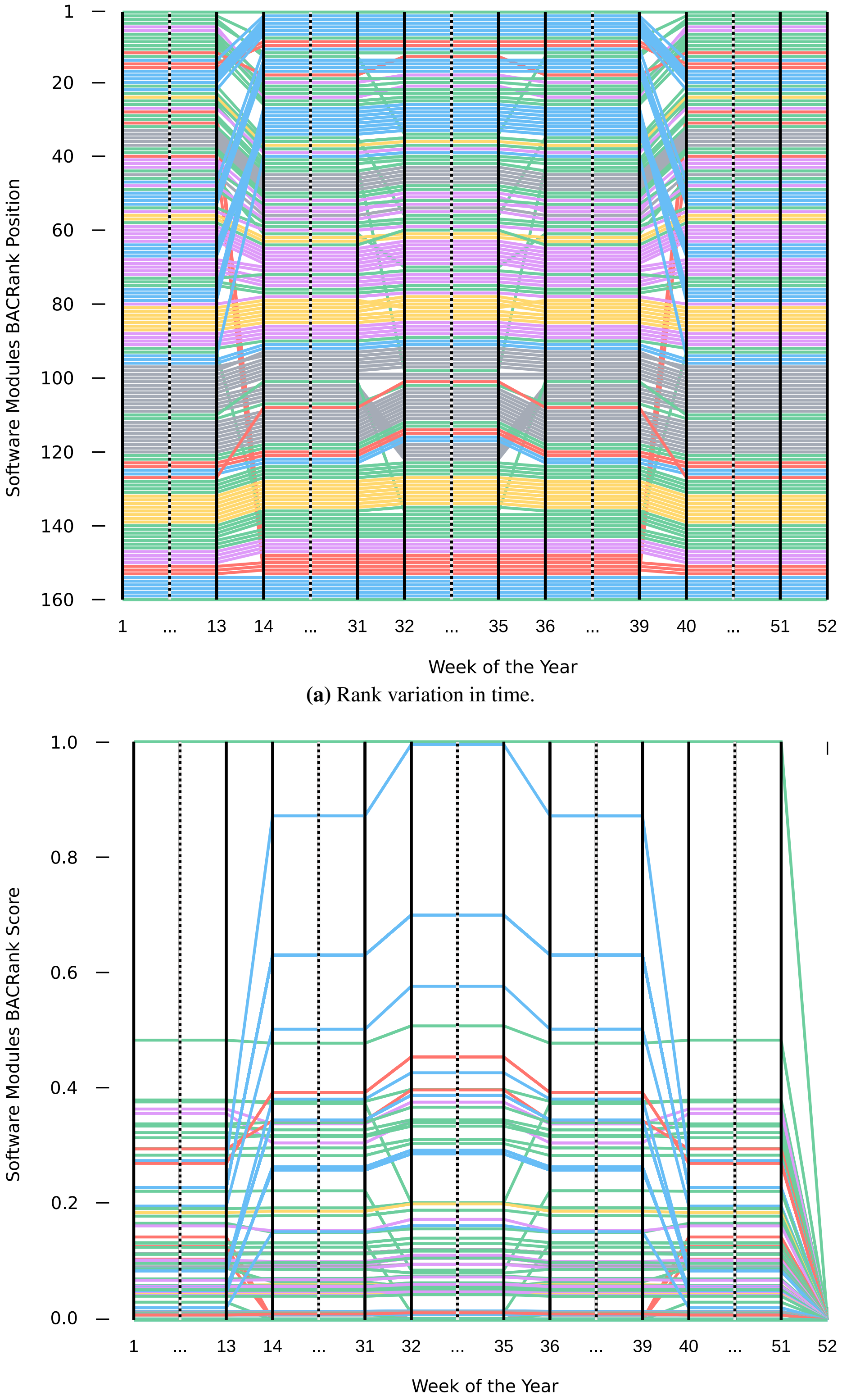


Fig. 4. BACRank Scores Convergence.

Time-*independent* software modules are typically ranked in similar positions throughout the year. Figure 5a shows that lighting and thermostat modules (yellow and gray lines) are good examples of time-independent modules. Their minor shifts up and down respond mostly to score variations in other modules rather than their own scores.

Time-*dependent* modules, on the other hand, are visible in Fig. 5a between weeks 14 and 39 of the year. This represents roughly the period between April and September, that is warmer than the range between October and March, taking into account the geographical location of the building. Figure 5a shows that most cooling modules increase their rank in this period whereas some heating modules suffer a substantial decrease (blue vs. red lines). Heating modules that remain similar or even increase their rank in the April–



(a) Rank variation in time.

(b) Score variation in time.

Fig. 5. Software modules impact variation in time. Plots only legible in color.

September period are benefited from neighboring cooling modules that got their rank increased. For example, heating modules “Radiatorgroup South” and “Radiatorgroup North” in positions 15 and 16 in Weeks 1–13, climbed to positions 9 and 10 in Weeks 14–39. These two modules are labeled as B and C in Fig. 3, which shows their proximity to cooling modules. Exactly 4 cooling modules —D, E, F, G — depend on B and C.

Weeks 32–35 of the year (August) are part of the organization’s summer break in which some of the business processes stop execution. Student-related processes (BP_5 and BP_{12}) do not run in this period, and therefore, the related software modules lower their ranking positions. Module “AHU WestLectRoom”, for example, decreases its rank from position 13 in week 31 to position 34 in week 32. The main reason for its descend is its support to the halted BP_5. This module is labeled with the letter H in Fig. 3.

In weeks 40–51 all software modules rank in the same order they ranked at the start of the year, due to identical conditions in terms of business processes running and software modules needed. Week 40 marks the start of the winter period in which cooling modules decrease their relevance in favor of heating modules as shown in Fig. 5a and Fig. 5b.

Finally, in week 52 the organization is closed and no business processes are running in this building. As explained before, Fig. 5a will simply assign an arbitrary order to equally ranked modules, whereas Fig. 5b shows that all the modules get a score of 0 which means that from the business continuity viewpoint all modules are “equally unimportant”.

6 Conclusion

We have presented the first BACS impact assessment methodology that is focused on business continuity. Our approach takes into account business and technical aspects from diverse information sources. The proposed methodology scores BACS software modules considering their support to the related business processes and their relevance to other neighboring modules.

Since software modules constitute a dependency graph, our methodology to score modules is modeled as a graph centrality measure. We formally defined the general requirements that such centrality measure must satisfy to give scores that reflect the modules’ relevance in the BACS infrastructure. Finally, we developed one instance of such centrality measure, which we called *BACRank*. We formally proved that BACRank satisfies the defined general requirements and evaluated it in a real BACS. The evaluation showed that BACRank successfully prioritizes the most relevant software modules with respect to the business continuity perspective.

Our comprehensive scoring methodology provides valuable insights about the BACS infrastructure typically overlooked by building administrators. Module dependencies, for example, might organically grow as the BACS evolves to the point in which administrators are no longer fully aware of the role they play and their overall impact in case of failures or active attacks.

Acknowledgments. This work is partially funded by the Costa Rica Institute of Technology. The authors would like to thank Henk Hobbelink and Lisseth Galán-Calderón for their help throughout this project.

References

1. Al-Mhiqani, M., Ahmad, R., Yassin, W., Hassan, A., Abidin, Z., Ali, N., Abdulkareem, K.: Cyber-security incidents: a review cases in cyber-physical systems. IJACSA **9**(1) (2018)
2. ANSI/ASHRAE STANDARD 135-2016: A data communication protocol for building automation and control networks (2016)
3. ANSI/ASHRAE STANDARD 188-2018: Legionellosis: Risk management for building water systems (2018)
4. ANSI/ASHRAE STANDARD 62.1-2016: Ventilation for acceptable indoor air quality (2016)
5. ANSI/ASHRAE STANDARD 62.2-2016: Ventilation and acceptable indoor air quality in residential buildings (2016)
6. Béla, G., István, K., Piroska, H.: A system dynamics approach for assessing the impact of cyber attacks on critical infrastructures. IJCIP **10**, 3–17 (2015)
7. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems **30**(1-7), 107–117 (1998)
8. Cárdenas, A., Amin, S., Lin, Z., Huang, Y., Huang, C., Sastry, S.: Attacks against process control systems: risk assessment, detection, and response. In: ASIACCS 2011. ACM (2011)
9. Cheminod, M., Durante, L., Valenzano, A.: Review of security issues in industrial networks. IEEE Trans. on Industrial Informatics **9**(1), 277–293 (2013)
10. Corzine, S.: Operational and Business Continuity Planning for Prolonged Airport Disruptions, vol. 93. Transportation Research Board (2013)
11. Esquivel-Vargas, H., Caselli, M., Peter, A.: Automatic deployment of specification-based intrusion detection in the BACnet protocol. In: CPS-SPC 2017. pp. 25–36. ACM (2017)
12. Fauri, D., Kapsalakis, M., dos Santos, D.R., Costante, E., den Hartog, J., Etalle, S.: Leveraging semantics for actionable intrusion detection in building automation systems. In: CRITIS 2018. pp. 113–125. Springer (2018)
13. ISO, BS: 27031: 2011. Information technology –Security techniques– Guidelines for information and communication technology readiness for business continuity. BSI (2011)
14. ISO, BS: 22301: 2012. Societal security. Business continuity management systems. Requirements. BSI (2012)
15. Krammer, L., Kastner, W., Sauter, T.: A generic dependability layer for building automation networks. In: WFCS 2016. pp. 1–4. IEEE (2016)
16. Krotofil, M., Cárdenas, A.: Resilience of process control systems to cyber-physical attacks. In: NordSec 2013. pp. 166–182. LNCS 8208, Springer (2013)
17. Li, X., Zhou, C., Tian, Y., Xiong, N., Qin, Y.: Asset-based dynamic impact assessment of cyberattacks for risk analysis in industrial control systems. IEEE Trans. Industrial Informatics **14**(2), 608–618 (2018)
18. Market Research Future: Building automation system market research report - global forecast to 2022 (2019)
19. Merz, H., Hansemann, T., Hübner, C.: Building automation. Springer (2009)
20. Orojloo, H., Azgomi, M.A.: A method for evaluating the consequence propagation of security attacks in cyber-physical systems. Future Generation Comp. Syst. **67**, 57–71 (2017)
21. Shirey, R.: Internet Security Glossary, Version 2. RFC 4949, IETF (2007)
22. World Health Organization and Pan American Health Organization: Hospital safety index: Guide for evaluators (2nd edition) (2015)
23. Yang, C., Shen, W., Chen, Q., Gunay, B.: A practical solution for HVAC prognostics: Failure mode and effects analysis in building maintenance. J. of Building Engineering **15** (2018)
24. Zeng, W., Zhang, Y., Chow, M.: Resilient distributed energy management subject to unexpected misbehaving generation units. IEEE Trans. on Industrial Informatics **13**(1) (2017)
25. Zimmermann, T., Nagappan, N.: Predicting defects using network analysis on dependency graphs. In: ICSE 2008. pp. 531–540. ACM (2008)