Research in Engineering Design (2023) 34:421–442
https://doi.org/10.1007/s00163-023-00417-3

ORIGINAL PAPER

Product representation via networks methodology for exposing project risks

Shlomi Efrati[1] · Yoram Reich[1]

Received: 26 January 2022 / Revised: 1 October 2022 / Accepted: 28 February 2023 / Published online: 13 April 2023

Abstract
One of the significant factors in the time to market of a technology-based product development project is effective risk management. Both the system engineer and the project manager must work together to map and manage the risks in the project throughout its lifetime. Risks in the project arise from various reasons, which are not necessarily quantifiable, but all of which must be managed by the project team. We propose a methodology for calculating the risk level origin in each system element or component, taking into account its role within the system containing these elements and its availability in the project timeline. This risk level can be used as an additional decision support tool for the project stakeholders. For this purpose, we present a four-step process for (1) graph network mapping of products, (2) applying network algorithms, (3) weighting with information from the project management discipline, and (4) calculating risk index for identifying risks. The resulting level of risk index will enable the project team to map and manage efficiently and effectively the risks arising from the system components throughout the life of the development project. To demonstrate the methodology, we analyzed two products from different fields and at different levels of abstraction. We derived from each case the risk index for the use of the project personnel.

Keywords Network graph · System engineer · Project manager · Product design · Risk management · Degree centrality · Betweenness centrality

1 Research motivation

A technological product development process includes fundamental steps in any design methodology: requirements analysis, product design, execution, and testing. The Vee model (Blanchard and Blyler 2016) defines these main life cycle phases as decomposition and definition, implementation, integration, and recomposition. This development process includes two main functions, which are present throughout the life cycle: project management and system engineering, which are assigned to a person or a group of people and allow separation between the administrative aspect (project management) and the engineering aspect (system engineering). Even in cases where there is no official assignment to one or another position, in practice, the responsibilities of these positions are spread among the project team.

By its very essence, a development project is an attempt to create something new and therefore includes an element of risk, an uncertain event that can affect the outcome. The project risks must be identified and managed during all stages. Since their range is vast, their sources are spread over many disciplines, starting with pure engineering, through psychology, and ending with a force majeure. As a result, the field of risk management in projects should consider many areas while ensuring the reduction of threats and increasing opportunities to achieve the ultimate goal—the project's success.

Scholars divide the risk management process into four stages: identification, analysis, planning, and execution. These activities are led by the two functions disciplines above: project management and system engineering. The risk assessment is based on the probability (of the existence of events) and originates from evaluations coming from the

✉ Yoram Reich
yoramr@tauex.tau.ac.il

[1] School of Mechanical Engineering and System Engineering Research Initiative (TAU-SERI), Tel Aviv University, Tel Aviv, Israel

engineering and administrative disciplines. Risk assessment and presentation tools are many and varied; each project team or organization chooses the appropriate one.

From personal observations, we realized that system engineering and project management have different tools to carry out their ongoing assignments, which contain, each in its field, important information regarding the development project. If we could manage to examine the information stored in these tools and analyze it, we will be able to derive from this fusion an evaluation tool that can be used as decision support for the project team and the organization's leaders regarding the project's risk level.

From the architecture structure of the product, which includes the hardware (HW) and the software (SW), we produce a network diagram on which we run mathematical algorithms to find the influence of the central components of the system. From the time management tool, we use the availability of those components. These data are analyzed to produce a quantitative risk index for each system component according to its location in the system and its readiness.

2 Literature review

This literature review addresses numerous issues necessary for developing our proposed approach: the complexity of products, key disciplines involved in the product development process, product architecture, graph networks as a model of systems, and tools used for risk management.

The complexity of products is increasing rapidly, mainly due to business aspects, including customer requirements and competition in the market (Bencherif and Mouss 2020; Eppinger and Ulrich 2015; Kleinsmann et al. 2010; Yassine and Braha 2003). This complexity is featured in many forms, including functionality, the level of integration, meeting environmental conditions and regulations, and shape (Danilovic and Browning 2007; Genta et al. 2014; Publishing 2011). These constraints significantly increase the chance of project failures and encourage the development of new methodologies to effectively manage product development with control and risk management (Conchir 2010).

A classic example is smartphone development, which requires knowledge in engineering disciplines and beyond, such as psychology (Kleinsmann et al. 2010). Another example is the car, which is supposed to meet stringent safety requirements, and customer desires while maintaining an attractive price tag and reasonable costs and maintenance efforts.

To meet product development goals, companies and organizations operate according to an orderly development methodology that includes structured development processes depending on the nature of the product and the market (Pahl and Beitz 1996; Eppinger and Ulrich 2015; Patil et al. 2017; Yassine 2004).

Two main disciplines that are intertwined in product development are system engineering and project management (Sage and Rouse 2014). In complex product development projects, each domain will be staffed with one person or more, depending on the project's size and nature. In simple projects, one person may fulfill both roles in addition to other tasks (Locatelli et al. 2017). The system engineer represents the technical engineering aspect, while the project manager is in charge of the administrative one (Conchir 2010; Haskins et al. 2006; Kapurch 2010; Locatelli et al. 2017). There is a partial overlap between them (task definition, risk management, interface with the customer, etc.), and they must cooperate for the project's success (Kordova et al. 2019). Even though the system engineer and the project manager use different tools and methods to carry out their mission, both know the structure and features of the product they are developing collaboratively (Eppinger and Ulrich 2015).

A system architecture is a conceptual description showing the system's structure, its interaction with the outside world, and the combination of its various components to perform system tasks (Rechtin and Maier 2010). The presentation of the architecture can be in many forms, both visual and textual. Several disciplines, such as system engineering and enterprise engineering, use architecture description languages (ADLs) to describe system models or concepts (Dissaux et al. 2005). Hardware description languages (HDLs) like VHDL and Verilog are tools used by engineers to describe firmware systems at different levels of abstraction (LaMeres 2019).

Presenting a systems architecture visually is common and includes a wide variety of forms. The best known is a block diagram, in which a rectangle denotes each system element or subsystem, and arrows between the rectangles indicate the connections and flow directions between these elements (Harman and Dabney 2001; Nilsson and Riedel 2011). Another set of tools developed from software engineering and adopted by system engineering is the Unified Modeling Language (UML) diagrams of the Object Management Group (OMG). These diagrams are divided into groups according to a common characteristic. For example, structure diagrams (containing class diagrams and component diagrams) and behavioral diagrams (containing time diagrams and situation diagrams) (Booch et al. 1999).

It is also worth noting two common graphical tools for displaying the flow and structure of systems: the DFD (data flow diagram) and the IDEF0 (ICAM DEFinition for Function Modeling, where ICAM is: Integrated Computer Aided Manufacturing). The DFD graphically shows how the information flows within the system and interfaces with information sources outside the system. IDEF0, as part of the

IDEF family, is a modeling language that allows to analyze, develop and integrate systems (Dickerson and Mavris 2016).

The design structure matrix (DSM) and N2 chart are matrix representations of the relationship between system elements. These representations allow the user to analyze the flow and dependencies within the system and allow clustering to optimize elements organization (S. D. Eppinger and Browning 2012; Lano 1977). DSM can also be used to describe processes as well as the interactions between people (Eppinger and Salminen 2002). The DSM representation is compact and intuitive (Engel and Reich 2015; Eppinger and Browning 2012). On the other hand, the increase in matrix dimensions when adding more elements is rapid as well as the decrease in its readability (König et al. 2008).

In the field of software, there are several different architectures, from which the software architect or system engineer will choose to suit the application (Bass et al. 2003). The most common in generic applications installed on desktops is the layered architecture. It is suitable for a software structure that can be divided into groups of tasks, each of which can be assigned to a certain level of abstraction. Other notable software patterns are client–server; master–slave; pipe–filter; and peer-to-peer (Richards 2015).

Another tool, graph network from discrete mathematics, enables the mapping, presentation, and analysis of relationships between entities (Marcus 2008). Over the past 60 years, graph theory has become one of the fastest-growing mathematical areas (Gross and Yellen 2004) and is commonplace in many fields of research such as psychology, engineering, zoology, and cyber (Aleta and Moreno 2019; Newman 2010). Initially, attempts to describe these networks were based on probabilistic models designed to distinguish between families of networks and their characteristics. These principles were formulated in the 1960s by Erdos and Renyi and formed the basis of random graph theory (Erdos and Rényi 1960; Rényi 1959). At the end of the twentieth century, with the evolution of computers and the ability to collect and process big data, it became clear that many real-world systems were not operating according to the principles of random networks. Barabási and Albert (1999) laid the foundations for complex systems, which more reliably describe real-world systems and allow using an appropriate mathematical infrastructure to describe the features that characterize them.

The mathematical basis found in network science provides an additional layer for producing quantitative insights (Diestel 2000). Applying algorithms to a network allows information to be obtained on the entire network and discrete nodes. Indices such as density, diameter, scale, etc., are spatial ones, while centrality indices give the actual effect of a particular node in the network (Freeman 1977). Betweenness centrality measures how much influence a node has on the flow of the graph. Nodes that serve as bridges between different parts of a graph are often found in this method (Chen et al. 2018). The algorithm computes the shortest unweighted paths between all pairs of nodes. Each node receives a score based on the number of shortest paths that pass through it. The higher the betweenness centrality score for a node, the more frequently it is on the shortest path between other nodes (Brandes 2001). Degree centrality is a simple centrality measure that summarizes the number of edges entering or leaving (or both) a certain node. The higher the degree centrality index of a node, the more connected that node is considered (Newman 2010).

The essence of a directed network graph is to describe the flow between two nodes through the edges. A non-directed graph contains edges without arrows, which indicate an interface/connection between the connected nodes (Borgatti 2005; Hatala and George Lutta 2009; Wasserman et al. 1994). Scholars distinguish several types of interactions between two system elements: spatial interaction, such as mechanical contact between two parts of a system; Information flow, such as message transmission between a transmitter and receiver; material flow, such as fluid transfer between two components; energy transfer, such as between a voltage source and an electrical load (Engel and Reich 2015; Eppinger and Browning 2012; Pimmler and Eppinger 1994).

One of the disciplines that have taken advantage of complex network theory is product development or systems engineering. As one of the key issues in this field, product development is modeled through complex networks producing insights from parts of the process and the structure of the entire process (Braha 2016). Attempts to analyze networks of tasks in the product development process, both statistically and from other quantitative perspectives, showed similar patterns and characteristics to complex networks from other domains. Features like small-world, centrality metrics, robustness, performance, and flow improvement parameters were demonstrated in (Braha and Bar-Yam 2004b). The mirroring effect, which examines the relationship between product architecture and the structure of the organization, is largely also reflected through product development networks that include the organization's people (Braha and Bar-Yam 2004a). These results were reaffirmed in subsequent studies, including recently on product family design (Park and Kremer 2019).

Such studies demonstrate the pervasiveness of networks for addressing product development issues. More specifically, the combination of graph networks and reliability considerations can offer computational tools to product developers. (Cancela and Petingi 2004)offer an algorithm for calculating network reliability, assuming that the failure will be at the edges while the nodes do not fail (k-terminal reliability or classical reliability model). Additional measures for network reliability are the two-terminal and g-terminal approaches (Chaturvedi 2016). Given a network mapping of a product, graphical metrics can be used to calculate the importance of the system element in addition to its reliability (Cadini et al. 2009). Kurtoglu and Tumer (2008) offer a framework for assessing the risk of failure in systems and how it will spread within the system. It allows assessing the robustness of a given system and its behavior during a failure during the early stages of the system design.

Another engineering area, which is an extension of network theory, is complex networks (also known as multi-dimensional, multilayer, or multiplex networks) (Kivelä et al. 2014). This field of science allows the inclusion of different domains in a single-layer network to generate insights in multiple disciplines. Similar ideas can be found in multi-domain matrices (MDM) (König et al. 2008; Maurer and Lindemann 2008) or in the abstract world of systems description, such as the PSI (Reich and Subrahmanian 2020). A comprehensive review of system complexity can be found in (Summers and Shah 2010), during which discrimination was made between the problem, the process, and the product. In addition, a benchmark is proposed for evaluating that complexity level in three aspects: size, coupling, and solution ability (solvability).

On the administrative side of the product development project, two essential tools that are frequently used by the project manager can be noted: The Gantt and PERT chars (Conchir 2010). A PERT (Program Evaluation Review Technique) chart is a way of creating and displaying a project by showing tasks as boxes and the dependencies between tasks as lines between these boxes (Kerzner 2017; Lester 2014). A Gantt chart is a commonly used graphical depiction of a project schedule. It is a type of bar chart showing the start and finish dates of a project's elements, such as resources, planning, and dependencies (Kerzner 2017).

As part of their duties, the project managers also define and manage project risks (with the support of the system engineer). The term “risk” has multiple definitions, with no universally accepted one (Dorofee et al. 1996). We refer to project risk as a plausible event that is uncertain and may occur during the project life and affect its outcome (Hillson 2003; Kerzner 2017). According to (Hillson 2014), there are four types of risks: event risk, variability risk, ambiguity risk, and emergent risk. All of them should be monitored and managed. Project risk management is all the actions that are taken to ensure the project's outcome by reducing the threats and increasing the opportunities, which is divided into four stages: risk identification, risk analysis, risk response, and monitoring (Conchir 2010). More detailed stages were defined by the UK Association for Project Management which contains nine different phases for managing project risks (Simon 1997). Although risk management is something that must be done, not everyone performs it at all or correctly and effectively (Dorofee et al. 1996; Raz et al. 2002).

Practitioners are familiar with a phenomenon in product development processes in which development tasks are repeated over and over again. This phenomenon is cross-industry and is known as design churn – a lack of convergence in development activities during product development (Yassine et al. 2003). A project that features design churn can dramatically increase the risk of completion on time. Network or matrix representations (e.g., The DSM) of products or processes have been used to model design churn or development time, rework or change propagation (Clarkson et al. 2004; Sered and Reich 2003; Yassine et al. 2003), but they require extensive information. Further, no attempt has been made to combine product and process models (Browning et al. 2006).

Researchers make use of the visual and quantitative capabilities of graph networks in risk management. (Van den Brink et al. 2020)mapped Cobalt's global supply chain and demonstrated the risk hubs by finding the influential nodes in the network. At the organization level, mapping tools can be used to optimize product design processes to streamline the risk management process (Ahmadi and Wang 1999).

The tools that project managers use for mapping, quantifying, and managing risk in a project are diverse: brainstorming, interviews, checklist, root-cause analysis, Monte Carlo analysis, failure mode and effects analysis (FMEA), and risk matrices (Conchir 2010; Rausand and Hoyland 2003). The latter is a common tool that allows the risks to be presented across the plane and examined in the face of the project and other risks (Milosevic 2003). Aggregating risks and representing them with a single score is not trivial because the risk components are different: some are quantitative, and some are qualitative, in addition to their different impact on decision-making. At the same time, the decision-makers in the organization want to have one index that incorporates all the risk components (Li et al. 2015). Various approaches exist to risk aggregation, from the most conservative way of a simple summary to a workable framework that includes the use of mathematical and statistical tools (Bao et al. 2021).

The project risk management tools mentioned above, rely on past lessons or experience existing in the organization and deal with all possible risks. The methodology presented in this study suggests a tool for quantitative risk management, which weighs the role of the system element and its availability in the timeline of the project.

3 Product layer representation

3.1 Block diagram anatomy

We chose the block diagram as the primary tool for visual system architecture description from the methods

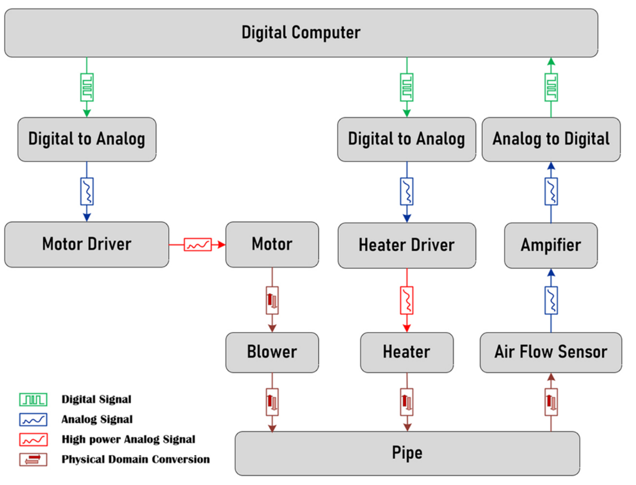


Fig. 1 Block diagram of airflow control system with analog/digital signal type and domain conversion flow indication

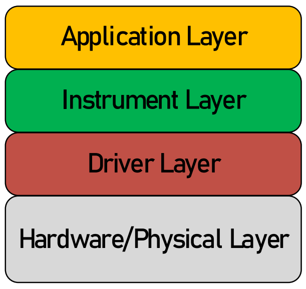


Fig. 2 The LabVIEW application system architecture (www.ni.com)

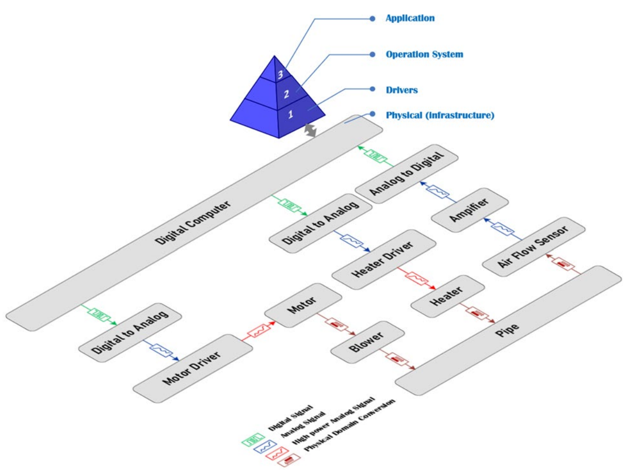


Fig. 3 The system from Fig. 1b with SW layers included

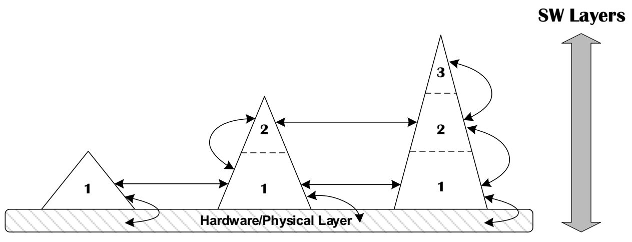


Fig. 4 The flow between SW layer inter and intra-programable components

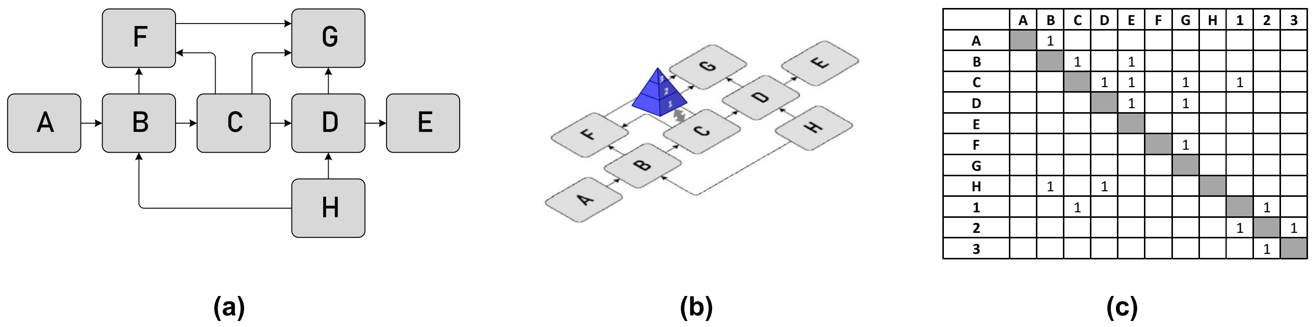
described above due to its popularity within the engineering community. In the most basic way, a block diagram consists of two main elements: a rectangle indicating a component or system function, and an arrow, indicating a relationship between two or more rectangles. This type of block diagram is common in describing systems: the rectangles indicate the function or system element, and the arrows describe the relationship and direction of flow (e.g., a functional diagram (Pahl and Beitz 1996)).

In multidisciplinary systems with more than one domain, we can add a layer of notation. Harnessing the principles from control theory, the distinction between the different relationships between system elements can be based on the nature of the relationship (Harman and Dabney 2001). For example, electrical domain versus physical domain (actuators/transducers); analog signal versus digital signal; high power signal versus low power signal; data line versus data bus, and so on. For example, Fig. 1 describes an airflow control system that contains system elements from different disciplines (sensors, actuators, processing units, etc.).

The "soft" elements of the system usually are not part of the block diagram and may have a separate description. Systems that contain programmable devices such as Field Programmable Gate Array (FPGA) with firmware (FW) code; microcontroller unit (MCU) with embedded code, and Central Processing Unit (CPU) with software (SW), should also be a part of the product description. For example, the SW architecture of the LabVIEW application, which is a systems engineering software for applications requiring testing, measurement, and control with rapid access to hardware and data insights, can be described as in Fig. 2.

The same layer structure can be found in Matlab/Simulink SW architecture (according to www.mathworks.com). The SW architecture can be adapted according to the subject application and the consideration of the system engineer. Combining both the block diagram, which represents the HW or the physical world, with the SW one is described in Fig. 3.

The relationships between different SW layers will be described according to the flow between them and other layers in other programming components in the system,



	A	B	C	D	E	F	G	H	1	2	3
A	▨	1									
B		▨	1		1						
C			▨	1	1		1		1		
D				▨	1		1				
E					▨						
F						▨	1				
G							▨				
H		1		1				▨			
1			1						▨	1	
2									1	▨	1
3										1	▨

Fig. 5 The transformation from product block diagram into adjacency matrix: **a** the HW block diagram; **b** adding SW elements to the block diagram; **c** the adjacency matrix, which reflects the system element connections

as shown in Fig. 4. The connections between the software layers are determined according to the software architecture and the interfaces between the hardware and the software. If we adopt the same connection principles between the different layers of the seven-layer model (Buchanan 2004), we can see that there is an interaction between adjacent layers and also between parallel layers between different network components. For example, the operating system layer is located between the application layer and the driver layer. It communicates with both and is therefore connected to them. In most cases, the application layer does not communicate directly with the driver layer. If so, they should be connected. If there are several computers in the system connected, there may also be connections between parallel layers (the application layer, the operating system layer, etc.). Note that in different systems, there may be other connections, which will be defined by the system engineer or the software architect.

Once we have defined the different representation and interconnection types for both hardware and software systems, we can use this notation as a basis for network graph representation. This representation will enable us to refer to all the system components at the same representation platform with all its associated benefits (see next section).

3.2 Harnessing graph networks to product representation

As mentioned above, a graph network is a tool whose roots are rooted in discrete mathematics, and in the last decades, its use has expanded to other fields. The graph network consists of two main elements: nodes and edges. In one abstract form, the nodes represent entities, and the edges represent the connections or interactions between nodes; these roles could also be reversed. Graph networks can be presented via an adjacency matrix.

Figure 5 describes the representation of a system architecture using an adjacency matrix. The physical layer (HW) described in Fig. 5a is augmented with the SW layer (Fig. 5b). Based on the above principles, we can formulate the adjacency matrix (Fig. 5c). A network graph derived from the adjacency matrix is depicted in Fig. 6a.

This resulting graph network describes the interconnections between the hardware and software components of the system. The nodes represent the components of the system, and the edges represent the connections and flow directions between them. The edges are unweighted, indicating an equal significant connection between two system elements. Since element C contains the software layers, they are linked to this node only.

Just from a visual impression in Fig. 6a, one can immediately identify the more influential nodes and those that are at the graph boundaries, and their contribution is marginal. If the graph contains separated networks as a result of discontinuity or error in the mapping, the viewer will be able to identify this immediately. In the quantitative aspect, we can take advantage of network algorithms to generate additional insights. Applying the degree centrality and betweenness centrality separately and independently on the original network yields the results shown in Fig. 6b, c, respectively. The outcomes of these centrality algorithms are numbers representing the weight of each node. The weight of the nodes is translated to the size of the radius of the node in the network graph to present this measure visually. The larger the node's radius, the larger is its centrality index.

Centrality measurements can demonstrate the influence that a node has according to various parameters. This form of expression allows the observer to graphically map the relative weight of a node in the network. The degree of centrality ranks the nodes according to the amount of connectivity in the network. In Fig. 6b, node C is the most linked node in the network. The betweenness centrality index indicates the weight of each node in terms of network or system flow. The higher the node's weight, the more significant the bridge flow is. Figure 6c indicates node C as the most significant node in system flow, while nodes like D, E, and G are less significant.

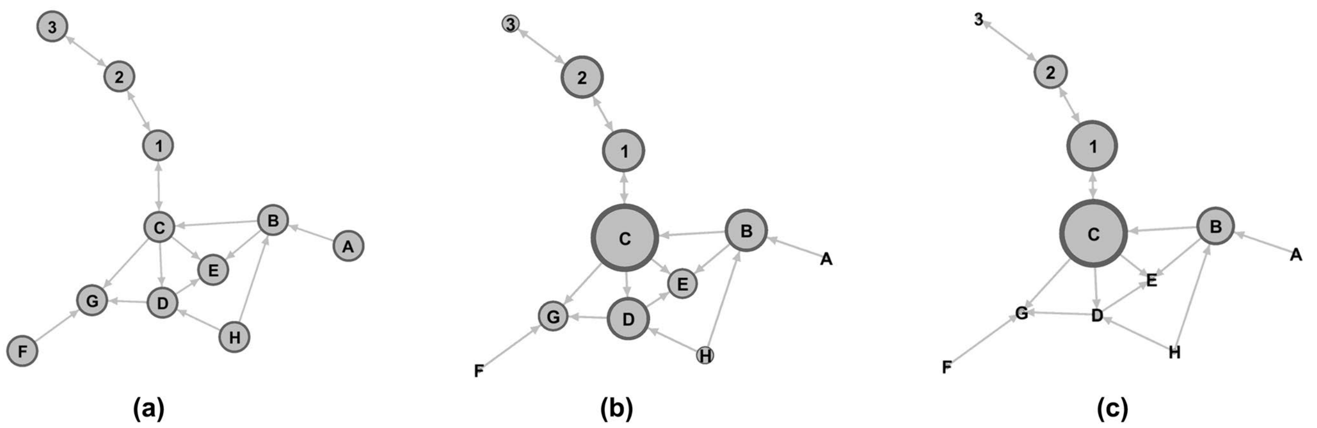


Fig. 6 The graph networks describe the system architecture in Fig. 5a; after applying the degree centrality algorithm (**b**); after applying the betweenness centrality algorithm (**c**)

3.3 Exploiting graph networks centrality measures for evaluating product elements influence

Representing systems using a network diagram is the first step in examining the effect of system elements on the entire system fabric. The system elements, represented by nodes, can be evaluated by applying network algorithms of centrality such as degree and betweenness. As mentioned above, the degree centrality index indicates the number of edges entering (in-degree) or outgoing (out-degree), or both (degree) to a given node. A node with a larger degree of centrality index has a greater number of connections. In our case, the degree centrality index indicates the number of interfaces that characterize a given node. The betweenness centrality index indicates the magnitude of the effect on the flow that a given node has in the graph. A larger betweenness centrality index for a node acts as a bridge connecting different parts of a graph. This index, which is applied to a system architecture represented by a network graph, will indicate the importance of the system element in the mediation of system parts. In other words, the extent to which a given node affects the system flow can be represented by the flow of information, energy, or matter.

In systems engineering processes, such as in design processes (according to the V-model (Blanchard and Blyler 2016)), these metrics can help the system engineer quantitatively assess the impact of system components on system aspects such as connectivity or impact level in system flow and compare against requirements or architecture. In the integration processes, the system engineer will have an additional point of view to estimate the design and optimization of the verification, validation, and testing (VVT) processes. For example, consider Fig. 7 depicting a system, represented by a network graph, that is characterized by two clusters connected by node B. The graph illustrating the degree centrality index is shown in Fig. 8, and the graph representing the betweenness centrality index is shown in Fig. 9.

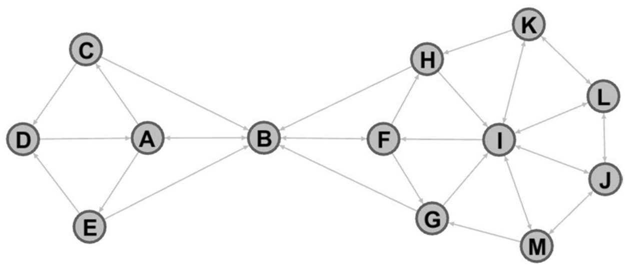


Fig. 7 System architecture representation via a network graph

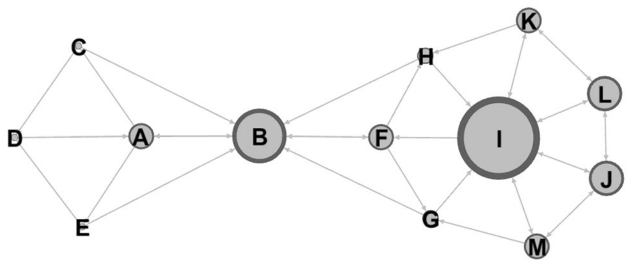


Fig. 8 The degree centrality of the network graph shown in Fig. 7

Without going into the numbering and from just observing the graph in Fig. 8, we can see that nodes I and B are the most linked at the interface level, while nodes like C and D are the less linked. The same goes for Fig. 9: node B is the most influential node in terms of system flow or bridge between the two clusters.

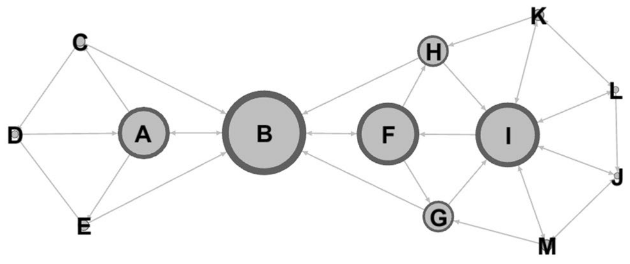


Fig. 9 The betweenness centrality of the network graph shown in Fig. 7

3.4 Project elements deliverable

The product development process is monitored and controlled by the project manager, who is responsible for managing the project team's tasks. To do this, the project manager uses task management tools such as Gantt or PERT charts. These charts contain significant information about the tasks themselves, who is in charge of each task, how long it takes to complete each task, and the dependencies between them. In addition, when using these tools, one can derive the supply time of the various system elements. Figure 10 depicts a Gantt chart for the system development shown in Fig. 5. The tasks are populated on the vertical axis, and the timeline is marked on the horizontal axis. The chart describes the different types of tasks, the work packages required to produce the various system elements,

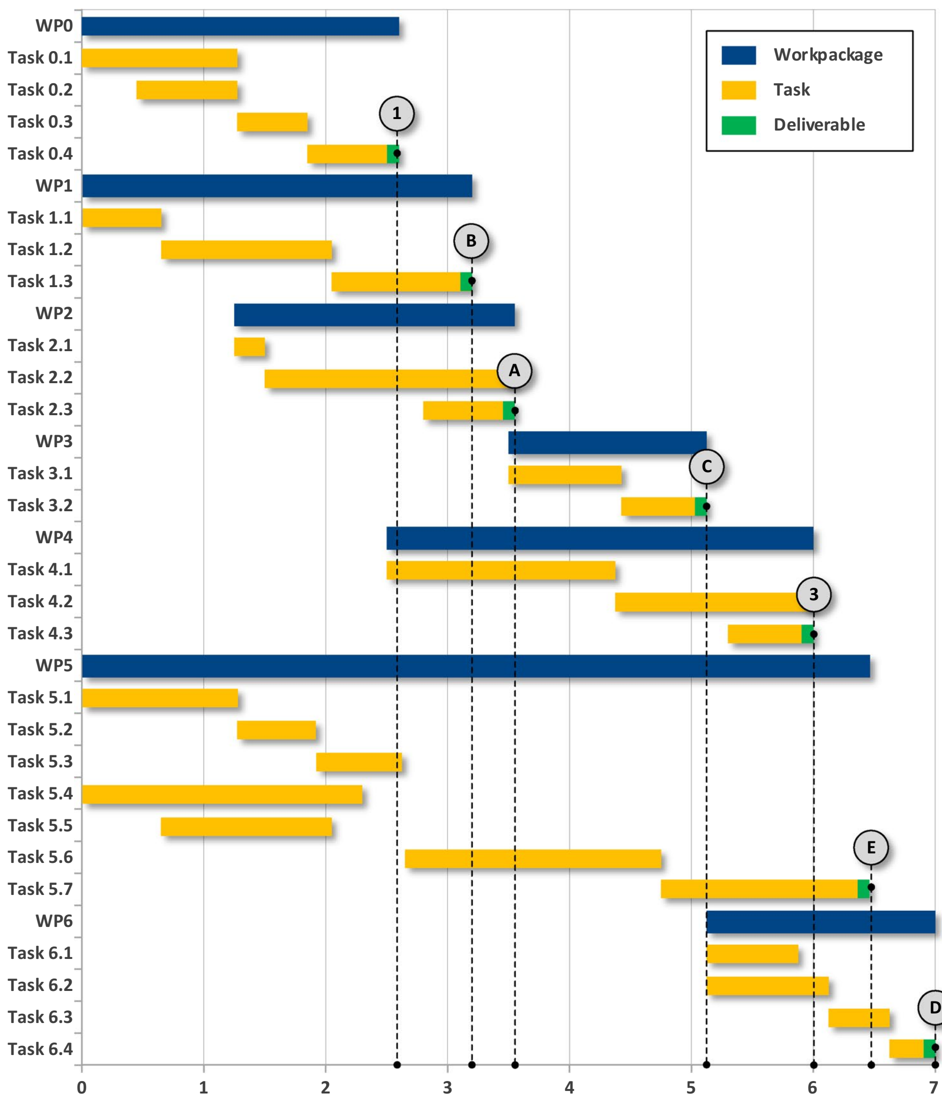


Fig. 10 A Gantt chart describing the project timeline (incl. deliverables) of the system shown in Fig. 5

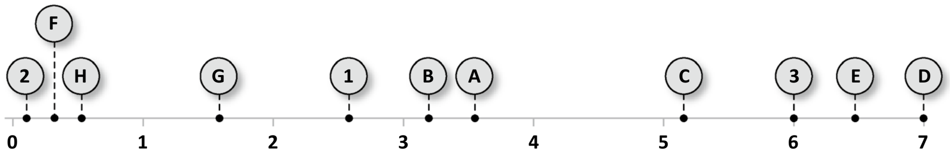


Fig. 11 System elements availability on the project timeline

and the expected delivery times of the elements. The time units that appear on the horizontal axis are generic and can be adjusted to any desired time scale (months, quarters, years, etc.).

To link the system components to the project diagram, we have specified the deliverable of each component in a circle. For example, the system component, E, will be ready at time 6.5. Please note that for the sake of convenience and simplicity of the diagram, not all the deliverable times of all the system components have been specified.

To distill the information from the Gantt chart and obtain the system components' availability, we can present the chart in Fig. 11.

3.5 Considering both product elements and product delivery time

To combine the information from the network mapping of the system in Fig. 6 and the schedules for the supply of its parts in Fig. 11, we plot them in Fig. 12 with the centrality index on the vertical axis and the timeline (or elapsed time) on the horizontal axis.

The charts in Fig. 12 depict different risk levels in the project. In chart (a), the elements of the system are described using two coordinates: one, indicates the component availability time in the project timeline, and the other, its weight in the degree centrality index. As said before, the degree centrality index describes the number of interfaces of a network node. In this case, the larger the vertical coordinate, the greater the number of its connections within the network. This means that as the system element is positioned to the right and up across the plane, it has a greater number of interfaces to other system components and will also be provided at later stages of the project. This poses at least a risk that if there is a problem with such and we will need to modify it, it can potentially impact numerous other components that were already delivered.

In chart (b), the vertical coordinate indicates the betweenness centrality index. This measure reflects the extent of the specific component as a bridge to system flows. In other words, as long as a system component is placed on a betweenness centrality–time plane to the right and up, it means that a significant component for flow (energy, information, matter) will be provided at later stages. This poses at least a risk that integration activities that test flows across the system will be postponed to late development stages. For the project management team, any one of these locations (whether on degree centrality–time or betweenness centrality planes) reflects risks that must be identified and managed early.

Note that the chart in Fig. 12 shows two independent indices: the delivery time of a system component and a centrality index. Delivery time is derived from the level of readiness of the subject component at the project level, while the centrality index is derived from the design architecture itself and the location and role of the component in the network. A component with high centrality metrics may be provided in the initial stages of the project due to reuse or procurement

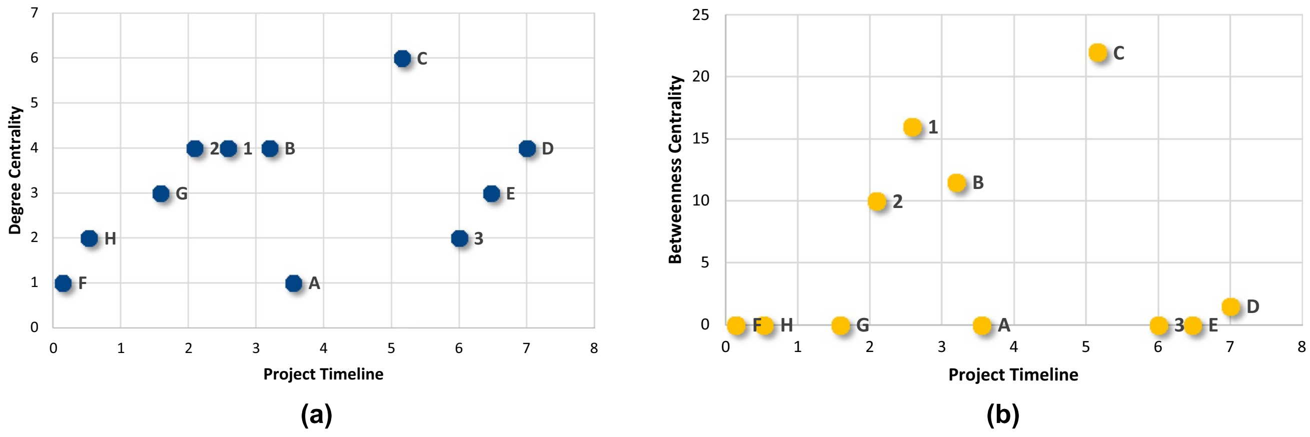


Fig. 12 Degree centrality vs. project timeline (**a**) and betweenness centrality vs. project timeline (**b**)

Fig. 13 Normalized project timeline, degree, and betweenness centrality measures on 1X1 chart

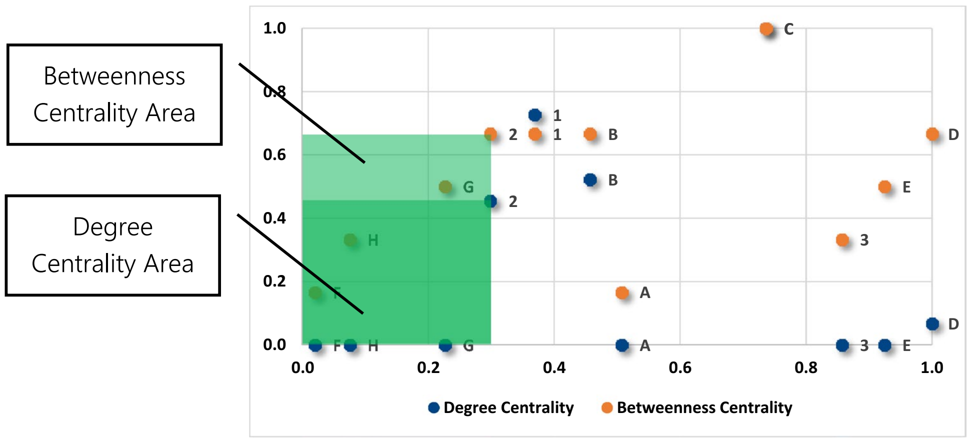


Table 1 The centrality measures and project deliverable data of the system network elements

Label	Betweenness centrality	Degree centrality	Time	Norn. BC	Norn. DC	Norn. time	Norn. BC*Norn. Time	Norn. DC*Norn. Time	Combined risk
A	0	1	3.5	0.00	0.17	0.5	0.00	0.08	0.08
B	11.5	4	3.2	0.52	0.67	0.5	0.24	0.30	0.54
C	22	6	5.2	1.00	1.00	0.7	0.74	0.74	1.47
E	0	3	6.5	0.00	0.50	0.9	0.00	0.46	0.46
D	1.5	4	7.0	0.07	0.67	1.0	0.07	0.67	0.73
G	0	3	1.6	0.00	0.50	0.2	0.00	0.11	0.11
1	16	4	2.6	0.73	0.67	0.4	0.27	0.25	0.51
F	0	1	0.1	0.00	0.17	0.0	0.00	0.00	0.00
H	0	2	0.5	0.00	0.33	0.1	0.00	0.03	0.03
2	10	4	2.1	0.45	0.67	0.3	0.14	0.20	0.33
3	0	2	6.0	0.00	0.33	0.9	0.00	0.29	0.29

from an external source. Pearson coefficients support the independence assertion: 0.35 for degree centrality vs. project timeline and 0.05 for betweenness centrality vs. project timeline.

To work with one combined index, we can calculate the areas that delimit each of the points and sum them up. The area marks a combined contribution of the two characteristics of a component. It is common in risk management tools such as FMEA to multiply characteristics to obtain their combined contribution. This way, we can get a normalized index for each of the risks. The first step is to normalize both centrality and timeline scales (Norm. BC/DC/Time) for compact visualization over a 1X1 chart (see Fig. 13). The second step is to calculate the areas under each coordinate (Norm. BC/DC* Norm. Time), as shown for element 2 in Fig. 13.

The last step is to sum for each component the area received from the degree centrality–time chart and the

Table 2 The system elements sorted by the DB–T factor

Label	DB–T factor	
C	1.47	High risk
D	0.73	
B	0.54	
1	0.51	
E	0.46	
2	0.33	
3	0.29	
G	0.11	
A	0.08	
H	0.03	
F	0.00	Low risk

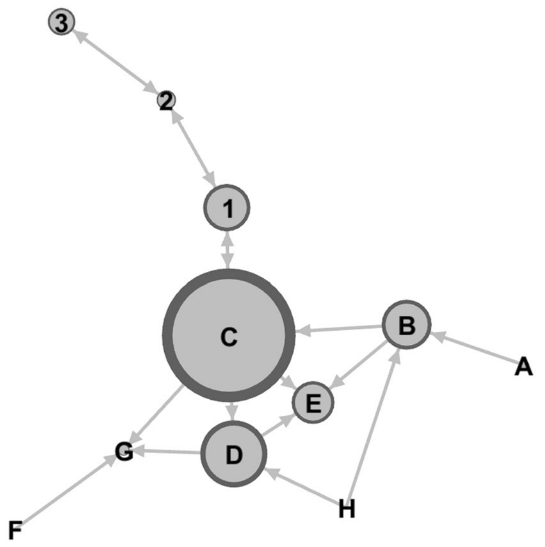


Fig. 14 The DB–T factor applied to the network graph shown in Fig. 7

betweenness centrality–time chart. The system component that demonstrates the highest value holds the highest combined risk, see Table 1. The combined risk can be referred to as degree betweenness–time (or DB–T) factor, and the sorted descending list (Table 2) is easy to monitor and control by the project manager and the other team members.

To visually present the weighted result of the risk index to the stakeholders, the same way of presenting the centrality indices can be used (see Fig. 14), only this time, the node size reflects the DB–T risk level. In this way, the viewer can immediately visualize the system element with the highest risk.

4 Implementation

To generate a network diagram of the product, according to the principles described above, we suggest the process in Fig. 15. The system engineer will provide information related to the product structure, its parts, and the connections between them. In addition, they deliver a block diagram or a similar document. We can convert the product's design to a network diagram, which consists of nodes representing the product's components and edges that show the connections between them according to the principles discussed above. Once the network mapping is obtained, it is possible to calculate degree and betweenness centralities, be impressed by the shape of the network, and produce qualitative insights. From the project manager, we can obtain data regarding the planned delivery dates of the system components. This will usually be achieved using Gantt and PERT charts, from which we can deduce when the system elements will be provided.

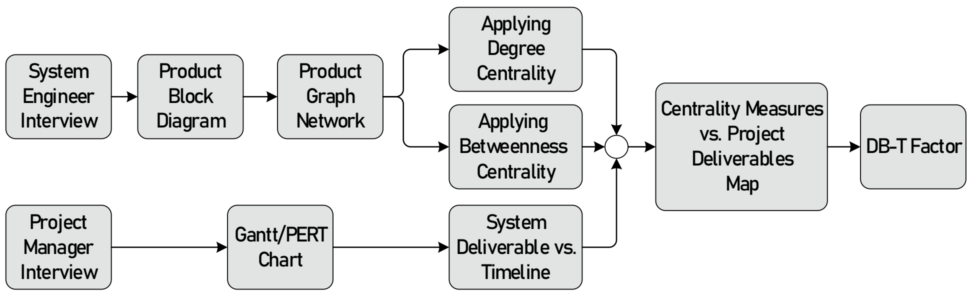
The two types of information, structural and logistical, can be summarized in a table and presented, as exemplified above, on a normalized 1X1 chart. According to the main criterion (DB–T factor), the risk levels of the various components will be compiled in one list that will serve as part of the set of different risks that have to be managed in the project.

The stages described in Fig. 15 can be implemented in a software environment. Since there is no such integrated software platform, the steps are executed manually. Software such as MS-Project (www.microsoft.com), in the field of project management, can be used to extract the project's Gantt chart. Software such as Gephi (www.gephi.org) can be used to build a network graph and calculate graph metrics.

Depending on the level of complexity of the system and depending on the level of abstraction discussed above, it is possible to choose whether to perform the analysis on the system as a whole or parts of it. Since product development is a dynamic process in which project and architectural aspects change, it is necessary to repeat the process each time to reflect the exact amount of risk.

Sensitivity analyses and "what-if?" scenarios, also characteristic of project execution, can use this methodology several times during the project lifetime to gain further insights. Assuming that the product architecture diagram and project schedules exist as part of the project required documents, the resources needed to perform the risk calculation according to this methodology are minimal, allowing to repeat it as the development project unfolds.

Fig. 15 Implementation process description


System Engineer Interview
Product Block Diagram
Product Graph Network
Applying Degree Centrality
Applying Betweenness Centrality
Project Manager Interview
Gantt/PERT Chart
System Deliverable vs. Timeline
Centrality Measures vs. Project Deliverables Map
DB-T Factor

5 Demonstration

This section will analyze two product architectures from different fields and at different levels of abstraction, represent them using network graphs, and derive risk insights from them according to the implementation shown in Sect. 4.

The first example shows an analysis of a driver for a medical motor which aims to demonstrate a relatively simple system, but still, one that includes several multidisciplinary elements connected to perform a common task. This system was designed by one of the authors, who accompanied the development of the project at all stages of its life cycle. The system's simplicity is intended to allow an understanding of the system components and their importance in fulfilling its objective.

The second example shows a more complex system. The objective of this use case is to demonstrate the analysis of a system of systems and to present important insights to the project manager and the system engineer while using the suggested framework.

5.1 Medical motor driver

Drivers for motors in medical applications must comply with some of the strictest standards since their failure may cause severe damage both to human life and equipment. Safety, reliability, and redundancy requirements are an integrated part of the specification documents defining this kind of instrument. The system engineer should consider all those requirements when determining the driver architecture on top of the electrical specifications related to the driver functionality.

A medical motor driver architecture used in an operational surgery robot is described in Fig. 16. The driver contains a power stage that interfaces with the motor and is controlled by a microcontroller and FPGA devices. The input command circuit determines the setpoint. The voltage and current sensor, as well as the temp. circuits are supporting mechanisms to monitor the proper operation of the system. The primary and secondary FPGAs work in parallel and execute the same tasks while monitoring each other. This redundant section of the system is part of the safety specifications. If there is a fault in one FPGA, the second one is responsible for alerting the user about it and shutting down the motor using a safe predefined procedure.

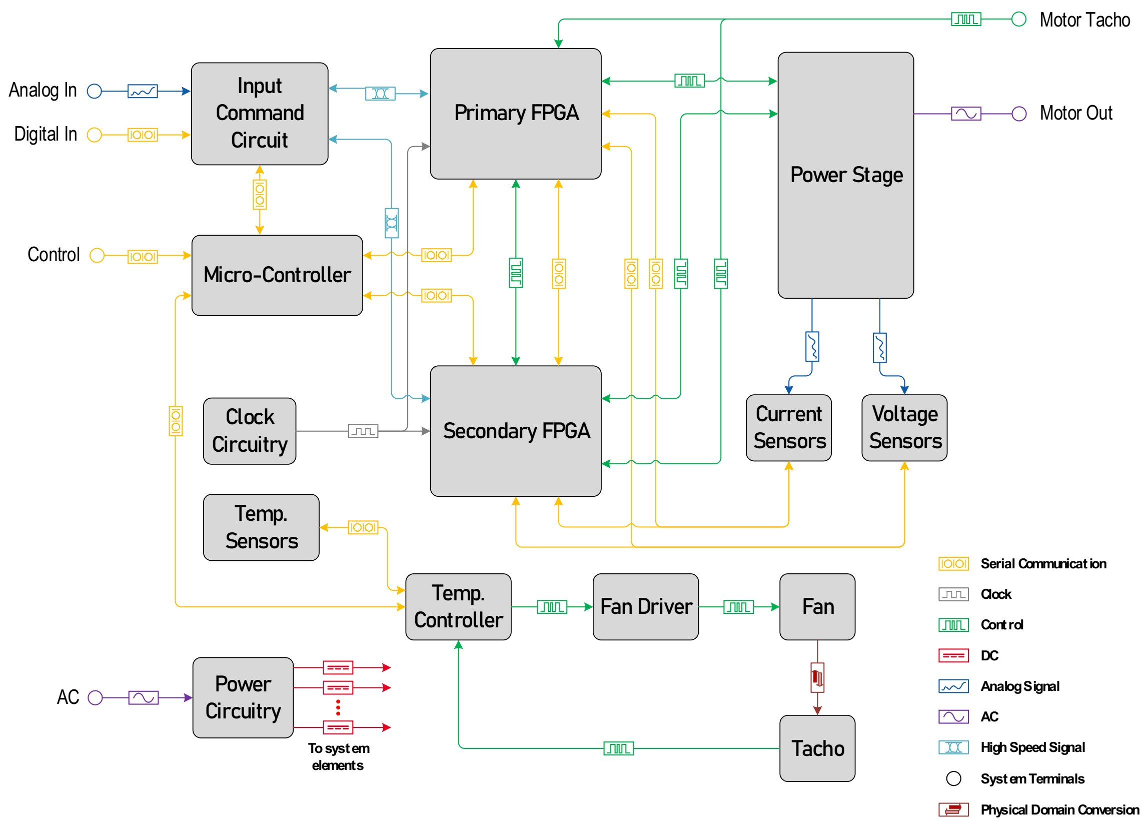


Fig. 16 Medical motor driver product architecture

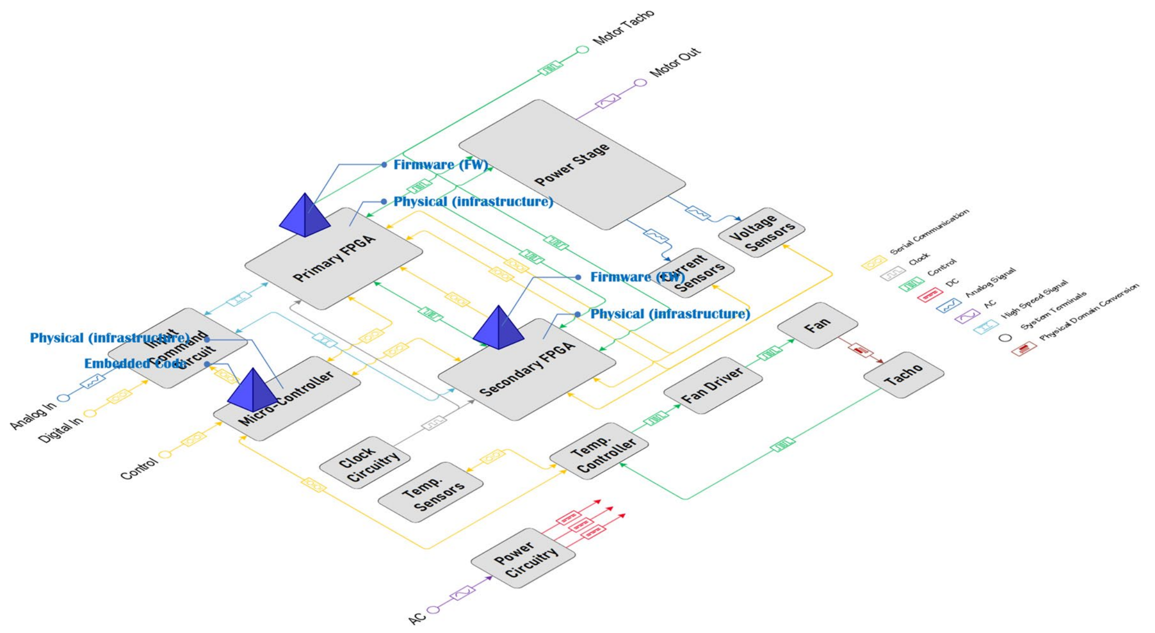


Fig. 17 SW components of the medical motor driver product

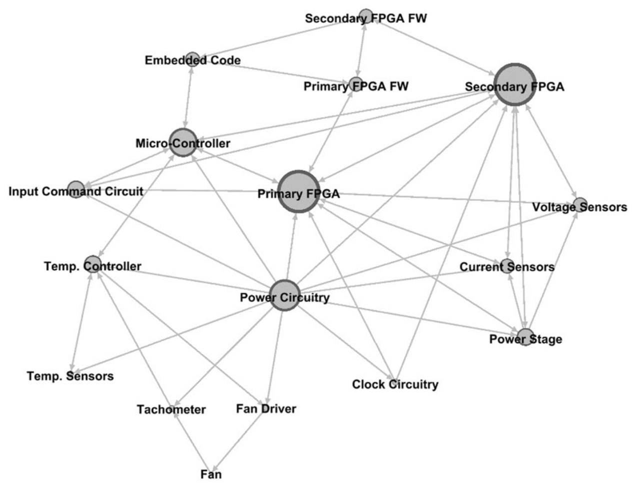


Fig. 18 Degree centrality of the system described in Fig. 16

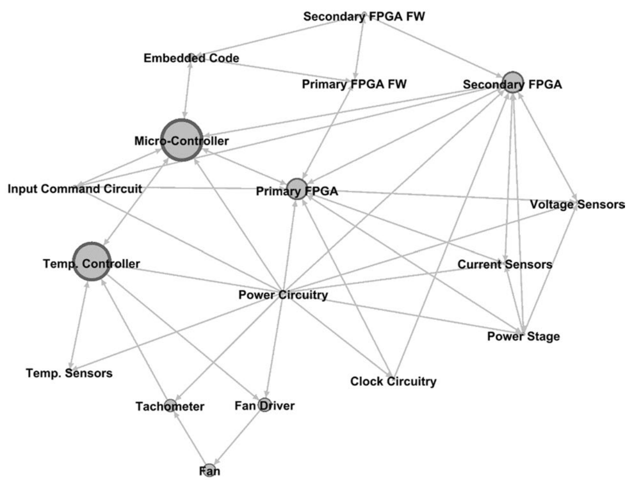


Fig. 19 Betweenness centrality of the system in Fig. 16

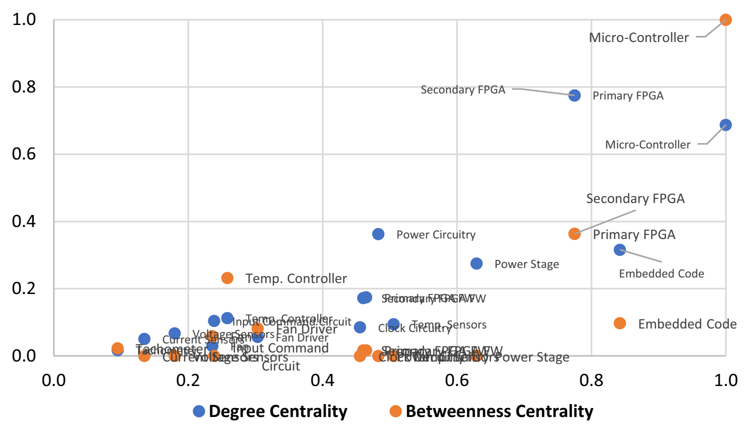
Please note that the output arrows from the power management block are connected to the relevant system elements, and for diagram clarity, we leave them unconnected.

The software aspect can be represented as described in Fig. 17. Modeling the above block diagram of the motor driver using a graph network and applying the degree centrality and betweenness centrality is shown in Figs. 18 and 19, respectively.

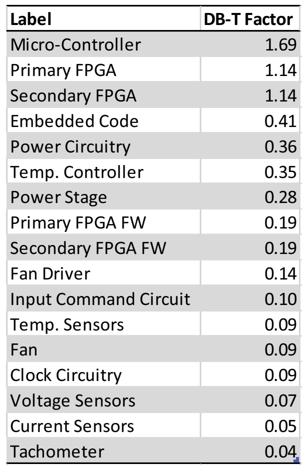
This symmetrical architecture is reflected in the above graph networks. One insight that can be driven from these graphs is the influence of the nodes reflected from the different algorithms. The temp. controller demonstrates minor influence in the interface connections (degree centrality), while in the system flow aspect, it has a great influence (betweenness centrality). Table 3 summarizes the system's centrality measures along with the system components' deliverable data. The 1X1 chart and the DB–T factor list are presented in Fig. 20. The aggregate risk obtained from both degree and betweenness centralities indicates that the microcontroller features the higher risk for the project. The temp. controller is less risky compared to the FPGA components. Key components in the system, such as controllers and FPGAs, are indeed at the top of the risk table. In case these components require new

Table 3 Summary table for the system in Fig. 16

Label	Betweenness centrality	Degree centrality	Time	Norm. BC	Norm. DC	Norm. Time	Norm. BC*Norm. Time	Norm. DC*Norm. Time	Combined risk
Power circuitry	0.00	12.00	4.30	0.00	0.75	0.48	0.00	0.36	0.36
Input command circuit	0.00	7.00	2.12	0.00	0.44	0.24	0.00	0.10	0.10
Microcontroller	103.50	11.00	8.90	1.00	0.69	1.00	1.00	0.69	1.69
Clock circuitry	0.00	3.00	4.06	0.00	0.19	0.46	0.00	0.09	0.09
Temp. Sensors	0.00	3.00	4.50	0.00	0.19	0.51	0.00	0.09	0.09
Primary FPGA	48.50	16.00	6.90	0.47	1.00	0.78	0.36	0.78	1.14
Secondary FPGA	48.50	16.00	6.90	0.47	1.00	0.78	0.36	0.78	1.14
Temp. Controller	93.00	7.00	8.00	0.90	0.44	0.26	0.23	0.11	0.35
Fan driver	28.00	3.00	2.70	0.27	0.19	0.30	0.08	0.06	0.14
Power stage	0.00	7.00	5.60	0.00	0.44	0.63	0.00	0.28	0.28
Current sensors	0.00	6.00	1.20	0.00	0.38	0.13	0.00	0.05	0.05
Voltage sensors	0.00	6.00	1.60	0.00	0.38	0.18	0.00	0.07	0.07
Tachometer	25.00	3.00	0.84	0.24	0.19	0.09	0.02	0.02	0.04
Embedded code	12.00	6.00	7.50	0.12	0.38	0.84	0.10	0.32	0.41
Primary FPGA FW	3.75	6.00	4.14	0.04	0.38	0.46	0.02	0.17	0.19
Secondary FPGA FW	3.75	6.00	4.10	0.04	0.38	0.46	0.02	0.17	0.19
Fan	26.00	2.00	2.10	0.25	0.13	0.24	0.06	0.03	0.09



(a)



Label	DB-T Factor
Micro-Controller	1.69
Primary FPGA	1.14
Secondary FPGA	1.14
Embedded Code	0.41
Power Circuitry	0.36
Temp. Controller	0.35
Power Stage	0.28
Primary FPGA FW	0.19
Secondary FPGA FW	0.19
Fan Driver	0.14
Input Command Circuit	0.10
Temp. Sensors	0.09
Fan	0.09
Clock Circuitry	0.09
Voltage Sensors	0.07
Current Sensors	0.05
Tachometer	0.04

(b)

Fig. 20 Medical motor driver 1X1 plane (**a**), and DB–T factor list (**b**)

design rounds, the impact on system integration will be significant. A graph network visually indicating the risk score is presented in Fig. 21.

5.2 Smart intersection system

Smart intersections, part of smart city infrastructure, enable optimal traffic management based on information from local sensors and spatial data sources. The ultimate objective of the system is to maintain the safety of road users while reducing the time required to travel from one location to the other. Many companies and enterprises (such as Swarco and Yunex Traffic) are harnessing the power of artificial intelligence (AI), fast communication, and high-performance power processing to smart cities and offering a complete solution for controlling and managing diverse and dense city

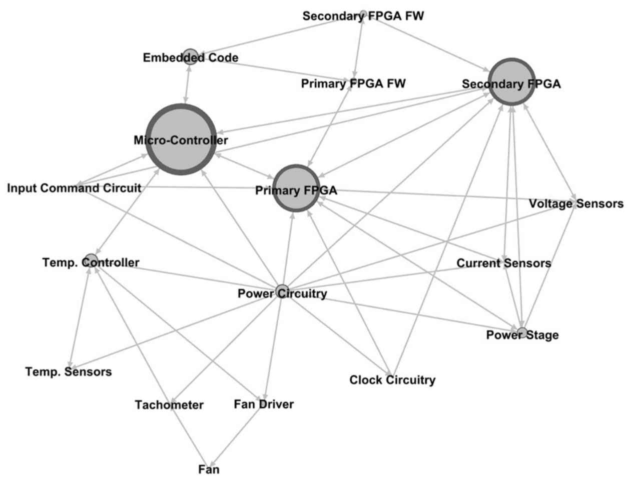


Fig. 21 The DB–T factor applied to the system described in Fig. 16

traffic. These systems are complex and can be referred to as systems of systems. A typical smart intersection system component is depicted in Fig. 22.

At the local road intersection and in addition to the familiar traffic light system, there is a layer of sensors including a proximity sensor, Cameras (optical sensors), and RADAR-based sensors, whose fusion gives the current traffic situation. A smart sign which presents relevant and updated information to the passing drivers is also a part of the smart intersection system and is used as a feedback channel from the control center.

The local controller, which manages the system at the intersection itself, is associated with a regional controller. Regional controllers run several intersections. The district controller, which contains an operations center, controls the district traffic. The system can be expanded to include higher levels of control hierarchies up to the national level.

Figure 23 describes a high-level block diagram of the smart infrastructure system with six intersections and hierarchies from the local intersection to the district level. The level of abstraction chosen is at the level of subsystems, which allows describing the system architecture without too much detail or being too abstract. The block diagram also contains a hierarchical description. The lower level is the intersection itself which contains the terminal components; the intermediate hierarchical level is the regional control which is a control hub, and at the higher level, there is the district control center. This architecture is modular and can be adapted to any size of intersections needed.

The software dimension presented in Fig. 24 includes different layers depending on the role of the programming device in the system and its position in the hierarchy. For example, programming components that interface with the system's end components include lower software layers, while computing parts in a higher hierarchy include working with human operators with higher software layers. The system architecture of a smart intersection using a network graph, which includes the hardware and software dimensions, is shown in Fig. 25.

A qualitative impression from the above graph network indicates that the terminal nodes include many interfaces, while in the center of the network, there are a limited number of central nodes that connect the terminal nodes. Applying the degree centrality and betweenness centrality is shown in Figs. 26 and 27, respectively.

The differences between the indices can be seen in the two illustrations above. Each controller is connected to many elements at the local level and therefore has a high degree centrality rank. In terms of system flow, which here is reduced to information flow only, the regional controllers are the ones that serve as bridges for the transfer of communication between the elements.

The summary table of the system described in Fig. 25, which contains the centrality measures along with the system components' deliverable data, is described in Table 4. The 1X1 chart and the DB–T factor list appear in Fig. 28. It can be seen from the DB–T index that two of the three components of the system that are at high risk are software elements. The component that is considered to have the highest risk is the local controller.

While in the previous example, the system was relatively simple, and one could guess the level of importance of the data nodes at the system level. In this complex case, the importance of the nodes at the system level is difficult to guess, and the network algorithms must be used. The results show that local controllers have less effect on the system flow level (low betweenness centrality index) but have many interfaces (high degree centrality index). The aggregation of these indices with the delivery times of the system components will make it possible to determine the level of risk posed to the project during the integration stages.

6 Discussion

Combining the centrality indices with project management data can produce valuable insights into understanding risks arising from the components of the technological product itself. Common risk management methods for system elements do not quantitatively discern each of the roles and influences. The above analysis makes it possible to rate the level of risk of each system element, taking into account both the level of its system influence and the date of its delivery or readiness on the project timeline.

From the medical motor driver system, we can learn that the microcontroller has the highest risk, according to the indices presented, followed by the FPGAs. Regarding

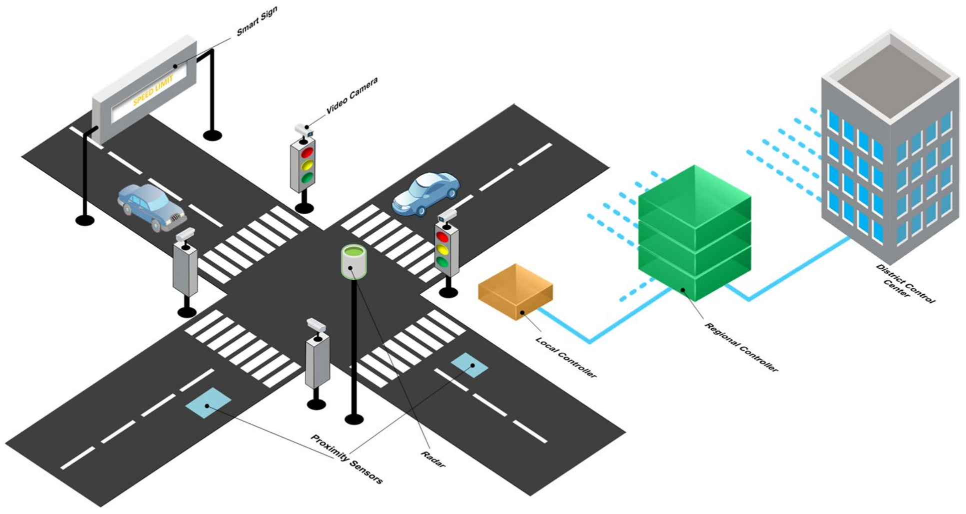


Fig. 22 Smart intersection system local and regional main components

the system architecture only, the temp. controller has a high betweenness centrality index and a medium degree centrality index. In weighing only the centrality indices, this component gains a higher place than the index that includes the time dimension. That is, considering the level of system importance only, we will get a different risk index, which may change if we augment the dimension of its availability in the virtual timeline. Given the above DB–T index, the project manager will be able to more optimally manage the risks arising from the system components by giving most of the attention to this critical component.

In networks that reflect complex systems or architecture of systems of systems, such as the smart intersection—finding the significant components will be done by algorithmics and less by intuition. In this case, the most significant risk components are the local processor, both at the hardware level and the software level. The ability to map all components of the system, both the hardware and the software, on the same scale and to execute metrics make it possible to reach integrative insights and not use superposition methods for unified insights. Applying the suggested methodology to this system makes it possible to rate the level of risk of each system component, even in the tangle of nodes and edges. In this case, also, the risky components due to the proposed approach are different from those employing only the centrality information and both may be different from those identified by common risk management approaches. Systems requiring a low level of risk would benefit from any new perspective on risk that is available.

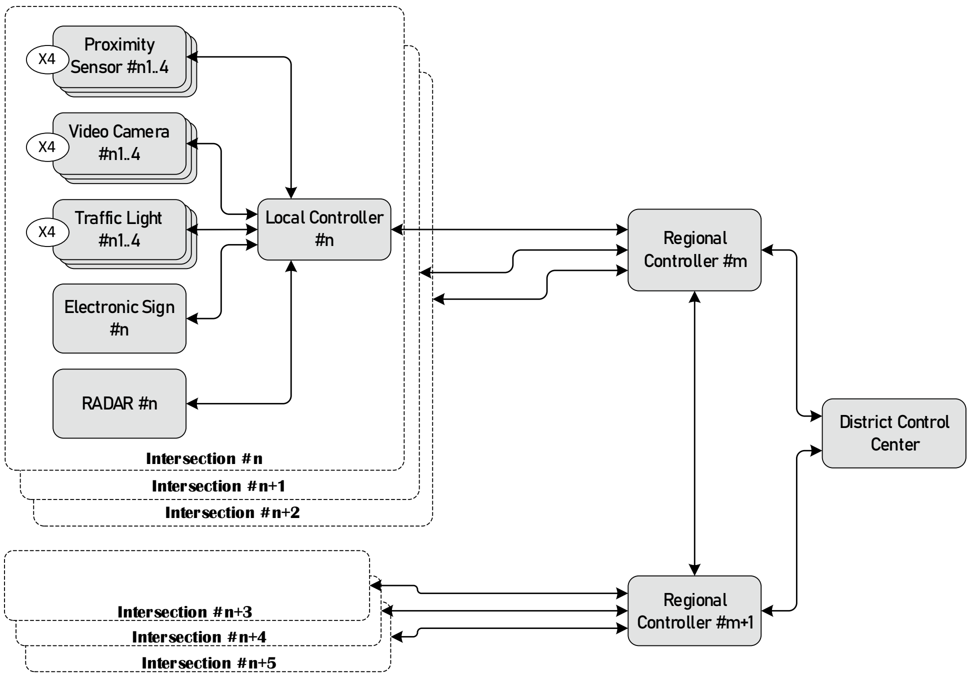


Fig. 23 Smart intersection high-level block diagram

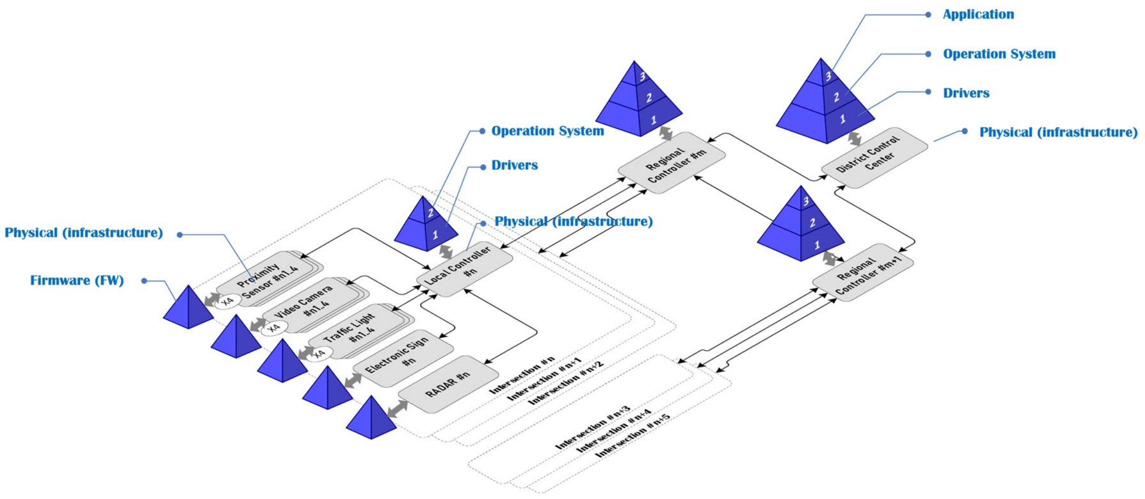


Fig. 24 Software dimension architecture of the smart intersection system described in Fig. 23

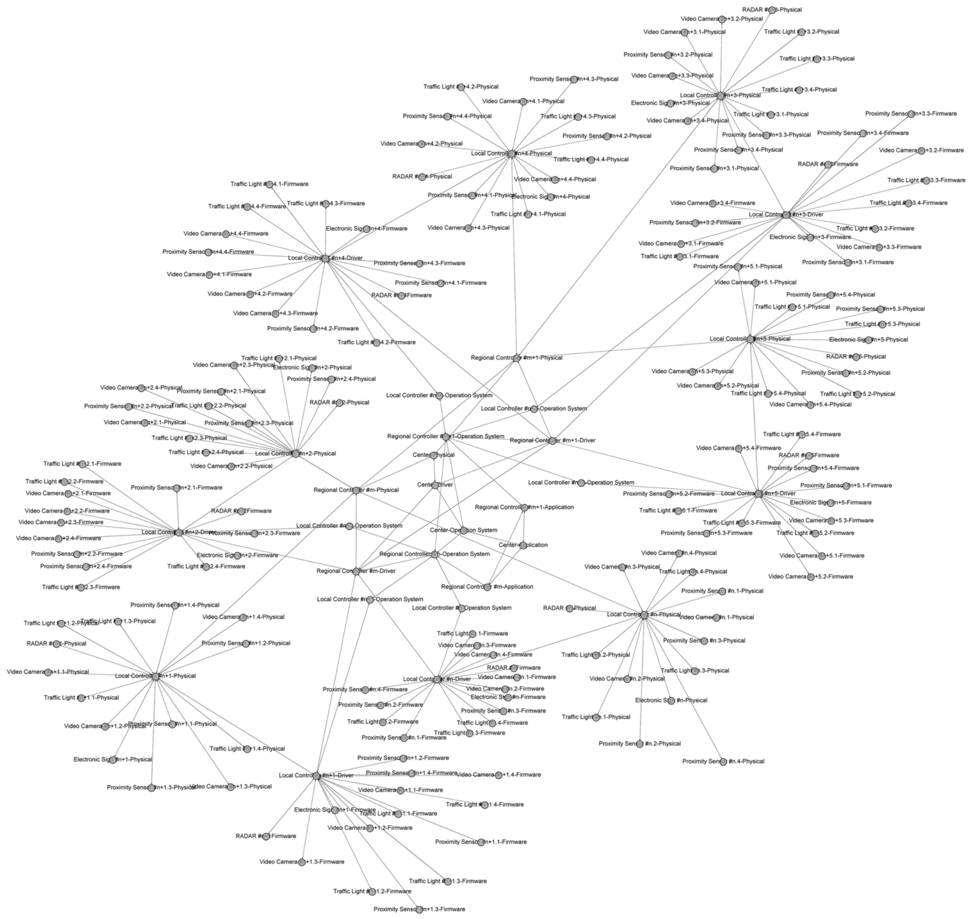


Fig. 25 Network graph of the smart intersection system described in Figs. 23 and 24

The model of the system as a network diagram reflects the hardware block diagram and software architecture. Any change in one of them (for example, in connectivity or elements) will be expressed in the network diagram and hence also on the results. This indicates the sensitivity of the model to reflect different modalities. Moreover, the project management team can update the model and derive results from it at the various stages of the project and thus further pinpoint the risk rating in the DB–T index.

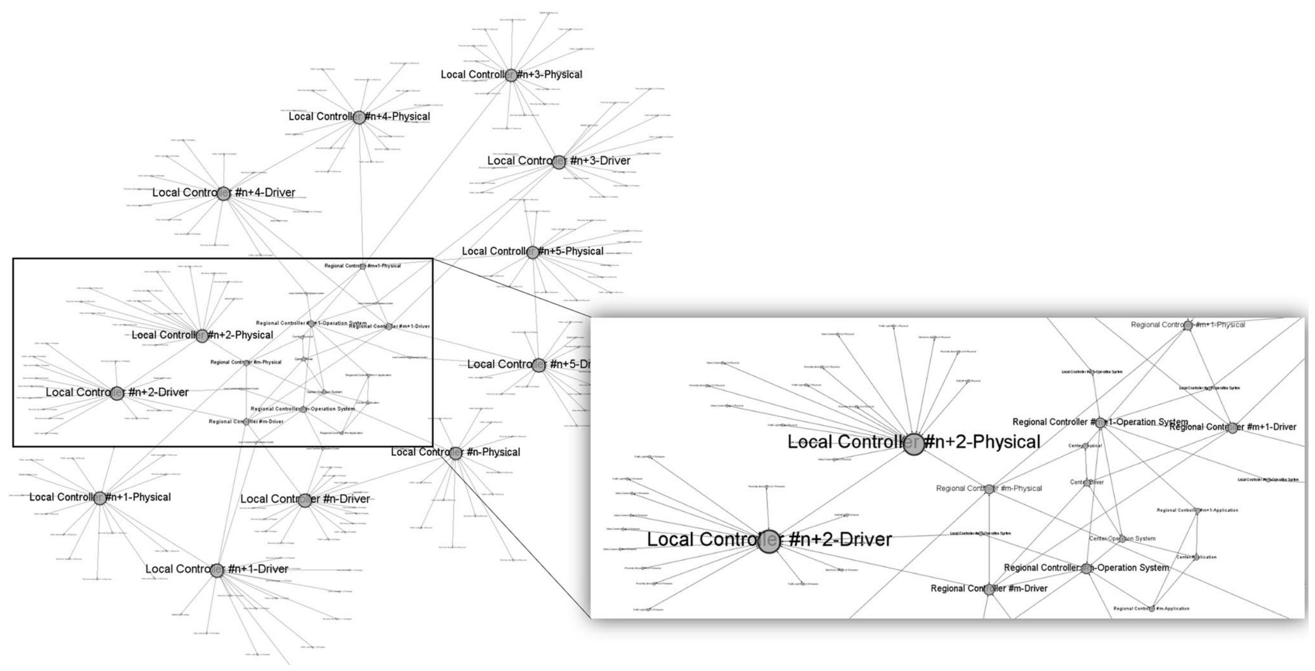

Local Controller #n+3-Physical
Local Controller #n+4-Physical
Local Controller #n+3-Driver
Local Controller #n+4-Driver
Local Controller #n+5-Physical
Local Controller #n+2-Physical
Local Controller #n+2-Driver
Local Controller #n+5-Driver
Local Controller #n-Physical
Local Controller #n+1-Physical
Local Controller #n-Driver
Local Controller #n+1-Driver
Regional Controller #m+1-Physical
Regional Controller #m+1-Operation System
Regional Controller #m+1-Driver
Regional Controller #m-Physical
Regional Controller #m-Operation System
Regional Controller #m-Driver

Fig. 26 Degree centrality of the graph network in Fig. 25

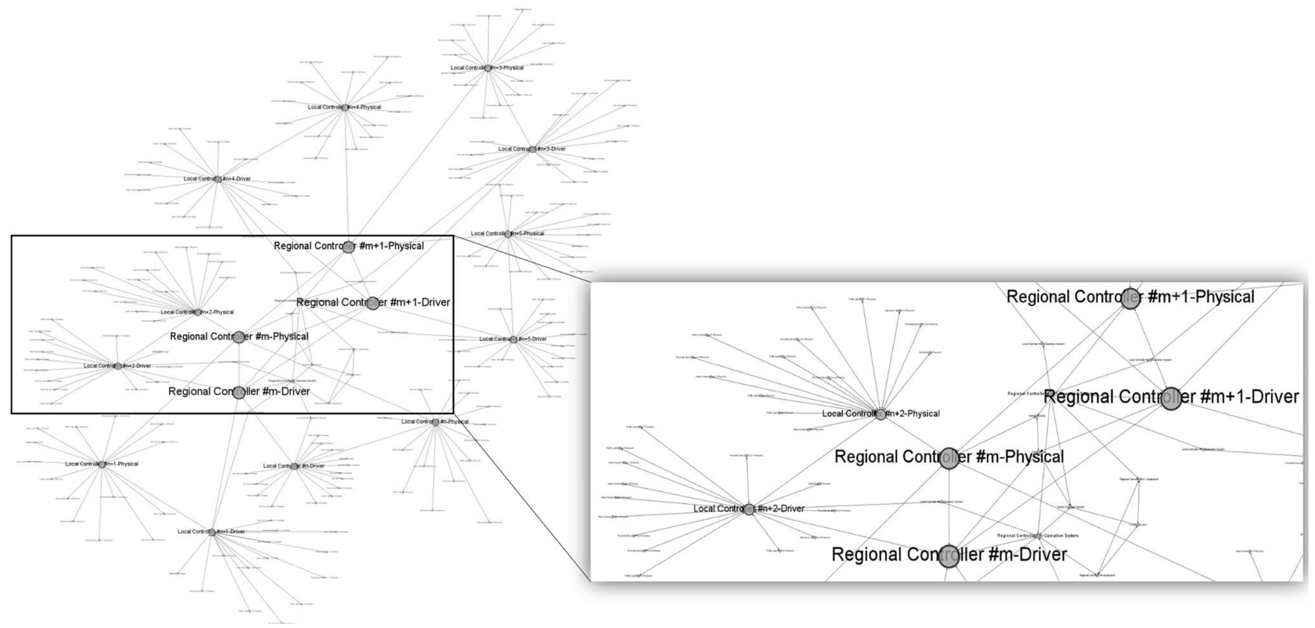

Local Controller #n+3-Physical
Local Controller #n+4-Physical
Local Controller #n+3-Driver
Local Controller #n+4-Driver
Local Controller #n+5-Physical
Regional Controller #m+1-Physical
Regional Controller #m+1-Driver
Local Controller #n+2-Physical
Regional Controller #m-Physical
Local Controller #n+2-Driver
Regional Controller #m-Driver
Local Controller #n+5-Driver
Local Controller #n-Physical
Local Controller #n+1-Physical
Local Controller #n-Driver
Local Controller #n+1-Driver

Fig. 27 Betweenness centrality of the graph network in Fig. 25

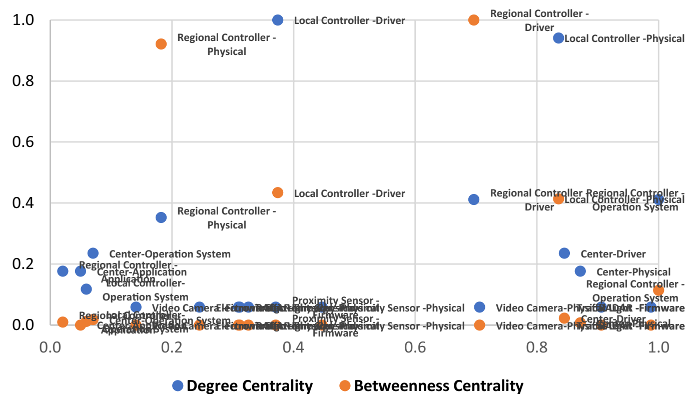
The analysis performed on the two test cases included hardware systems that integrate software. The common development model for hardware systems is the "Waterfall", while a common model for software development is the Agile model. So how, then, can they be combined into a single model? The reason for this is divided into two: the network model reflects the architecture of the hardware and software. In both the waterfall development method and the agile method, the architecture is usually stable and not subject to frequent changes. In the project management aspect, even if a software package is developed in an agile manner, there are milestones for the delivery of software packages

Table 4 Summary table for the system in Fig. 25

Label	Betweenness centrality	Degree centrality	Time	Norm. BC	Norm. DC	Norm. time	Norm. BC*Norm. Time	Norm. DC*Norm. Time	Combined risk
Center—application	4.33	6	4.7	0.00	0.18	0.0	0.00	0.01	0.009
Center—driver	331.67	8	80.9	0.02	0.24	0.8	0.02	0.20	0.218
Center—operation system	259.67	8	6.7	0.02	0.24	0.1	0.00	0.02	0.018
Center—physical	114.33	6	83.4	0.01	0.18	0.9	0.01	0.15	0.161
Electronic sign—firmware [#n..$n+5$]	0.00	2	23.5	0.00	0.06	0.2	0.00	0.01	0.014
Electronic sign—physical [#n..$n+5$]	0.00	2	31.2	0.00	0.06	0.3	0.00	0.02	0.019
Local controller—driver [#n..$n+5$]	6292.00	34	35.8	0.43	1.00	0.4	0.16	0.37	0.536
Local controller—operation system [#n..$n+5$]	178.83	4	5.7	0.01	0.12	0.1	0.00	0.01	0.008
Local controller—physical [#n..$n+5$]	5998.83	32	80.0	0.41	0.94	0.8	0.35	0.79	1.132
Proximity sensor—firmware [#n[1..4]..n+5[1..4]]	0.00	2	35.5	0.00	0.06	0.4	0.00	0.02	0.022
Proximity sensor physical [#n[1..4]..$n+5$[1..4]]	0.00	2	42.8	0.00	0.06	0.4	0.00	0.03	0.026
RADAR-firmware [#n..$n+5$]	0.00	2	86.8	0.00	0.06	0.9	0.00	0.05	0.053
RADAR—physical [#n..$n+5$]	0.00	2	29.8	0.00	0.06	0.3	0.00	0.02	0.018
Regional controller—application [#m..$m+1$]	142.33	6	2.0	0.01	0.18	0.0	0.00	0.00	0.004
Regional controller—driver [#m..$m+1$]	14502.67	14	66.7	1.00	0.41	0.7	0.70	0.29	0.983
Regional controller—operation system [#m..$m+1$]	1657.17	14	95.8	0.11	0.41	1.0	0.11	0.41	0.526
Regional controller—physical [#m..$m+1$]	13355.83	12	17.4	0.92	0.35	0.2	0.17	0.06	0.232
Traffic light—firmware [#n[1..4]..$n+5$[1..4]]	0.00	2	94.6	0.00	0.06	1.0	0.00	0.06	0.058
Traffic light—physical [#n[1..4]..$n+5$[1..4]]	0.00	2	29.6	0.00	0.06	0.3	0.00	0.02	0.018
Video camera—firmware [#n[1..4]..$n+5$[1..4]]	0.00	2	13.5	0.00	0.06	0.1	0.00	0.01	0.008
Video camera—physical [#n[1..4]..$n+5$[1..4]]	0.00	2	67.6	0.00	0.06	0.7	0.00	0.04	0.042

with known specifications. These are the points in time that the project manager will make sure to reflect in his project diagram and hence also for the calculation of the aforementioned risk index.

Considering the risk management process in the project, the project management team can use the methodology presented above to identify and analyze the risks arising from the role of the system elements and their readiness in the project timeline. The project team can use (as part of the response stage) its authority to mitigate or reduce the risks by taking actions related to a specific system element. For example, a project manager may decide to expedite production processes, start performing tasks earlier or allocate more resources to shortening task times to anticipate the delivery of a component at risk. Please note that it is not always possible to advance a task by allocating more



(a)

Label	DB-T Factor
Local Controller -Physical [#n..n+5]	1.132
Regional Controller -Driver [#m..m+1]	0.983
Local Controller -Driver [#n..n+5]	0.536
Regional Controller -Operation System [#m..m+1]	0.526
Regional Controller -Physical [#m..m+1]	0.232
Center-Driver	0.218
Center-Physical	0.161
Traffic Light -Firmware [#n[1..4]..n+5[1..4]]	0.058
RADAR -Firmware [#n..n+5]	0.053
Video Camera-Physical [#n[1..4]..n+5[1..4]]	0.042
Proximity Sensor -Physical [#n[1..4]..n+5[1..4]]	0.026
Proximity Sensor -Firmware [#n[1..4]..n+5[1..4]]	0.022
Electronic Sign -Physical [#n..n+5]	0.019
RADAR-Physical [#n..n+5]	0.018
Traffic Light -Physical [#n[1..4]..n+5[1..4]]	0.018
Center-Operation System	0.018
Electronic Sign -Firmware [#n..n+5]	0.014
Center-Application	0.009
Video Camera -Firmware [#n[1..4]..n+5[1..4]]	0.008
Local Controller-Operation System [#n..n+5]	0.008
Regional Controller -Application [#m..m+1]	0.004

(b)

Fig. 28 Smart traffic intersection 1X1 chart (**a**), and DB–T factor list (**b**)

resources, especially in cases of engineering design that cannot always linearly shorten its execution time (e.g., a project that includes the engineering design of a model lasting 40 h will not be shortened to 10 h if we assign four engineers to the task). The project manager, together with the system engineer, can find the sweet spots which allow optimization both at the architecture and at the project levels.

On the other hand, a system engineer can redesign the component and reduce the number of interfaces, provide simulation solutions or replacement modules for system integration and thus lower the uncertainty in the interface or system function. As an alternative step, the system engineer can also decide on an architectural change that will eventually lead to risk reduction. In other words, the system engineer is responsible for the displacement of the system elements on the 1X1 chart on the vertical axis, while the project manager is responsible for the displacement on the horizontal axis. Therefore, the risk management team consists of these two key personnel who must work in complete synchronization.

7 Conclusions

Risk assessment, taking into account both the system element and its readiness at the project level, can streamline technological project planning while meeting rapid time-market targets. The discussed methodology consists of several analytical tools from a variety of disciplines: system engineering, project management, and discrete mathematics. These tools, whose effectiveness has been proven many times in the past and are common in certain fields, were combined to enable a quantitative decision support tool for the project manager or system engineer in the product development process. We analyze a system structure according to the principles of abstraction and connectivity and present a network diagram consisting of hardware and software elements. This holistic description of a system using a network graph makes it possible to examine it according to tools from graph theory.

Applying network algorithms to the product graph network will enable the production of additional insights beyond the network topology reflected in the structure of the relationships between the elements. Centrality measures are significant for understanding the level of influence a node—or a system element—has in the overall network. From the degree centrality index, we can deduce how many interfaces each system element has. Subsequently, what do integration with this element involve, or how many system components affect a modification at the subject node? The betweenness centrality index makes it possible to determine the level of contribution of the subject node to the system flow—a quantity that includes the passage of information, material, or energy within the system's channels. The higher the level of influence of the node on the system flows, the more important it is for the proper operation of the system.

To perform a rating of risks arising from system components, we have to consider also some information from the project management: delivery dates or completion of development of the system elements. The time dimension augmented on the importance of the elements themselves allows, after some simple mathematical processing, to rank the components of the system according to the level of risk they pose. The described risk assessment methodology provides the project manager and the system engineer with a tool that can be used easily with information that could be updated according to the progress of the project.

We analyzed two systems in different levels of abstraction and fields to demonstrate the suggested methodology. The first example was a medical motor driver, and the second was a system array of smart interactions. For each system component, a DB–T index was calculated that weighs its level of risk while considering its function within the system and the relevant project data.

Please note that risk management in a project is a multidisciplinary field that contains logistical, engineering, psychological, probabilistic, and other elements. The tool proposed above is another mechanism to indicate the weight of each system element in the risk aspect of project success. Like a decision table, the DB–T index is also a tool for supporting decision-making and should be considered along with other risk management methods.

The above framework was demonstrated on two systems from different areas and levels of complexity. However, as part of this research, we did not validate it on a large number of designs and systems of different types, such as systems based on software/firmware only or firmware systems. To increase the validity of the suggested methodology, we intend to test it in additional product development projects of varying types.

Additional relevant research can evaluate other network measures, such as an analysis of node significance while considering weighted edges to reflect the importance of interfaces or flows or other centrality indices such as eigenvalue or closeness centralities. Another potential addition would be considering environmental influences or system level influences between system components due to implementation such as heat or electromagnetic influences. These may require different network or proximity modeling and its integration with the proposed approach seems valuable.

One can expand the study by considering the network graph to include additional layers of information beyond the product—such as processes and staff. Combining the network graph analysis methodology with other work frameworks, such as PSI, can yield insights even at levels beyond the product itself. It is worth suggesting a combined tool that will reflect the DB–T factor and additional known project risk for a more comprehensive risk assessment in the project management part.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00163-023-00417-3.

Data availability The data used in this paper is available as supplementary material.

References

Ahmadi R, Wang RH (1999) Managing development risk in product design processes. Oper Res 47(2):235–246

Aleta A, Moreno Y (2019) Multilayer networks in a nutshell. Annu Rev Condens Matter Phys 10(1):45–62. https://doi.org/10.1146/annurev-conmatphys-031218-013259

Bao C, Wan J, Wu D, Li J (2021) Aggregating risk matrices under a normative framework. J Risk Res 24(8):999–1015

Barabási A-L, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512

Bass L, Clements P, Kazman R (2003) Software architecture in practice. Addison-Wesley Professional

Bencherif F, Mouss LH (2020) Complex network to enhance characterization analysis in modelling product development process. Afr J Sci Technol Innov Dev 12(7):797–811. https://doi.org/10.1080/20421338.2020.1762355

Blanchard BS, Blyler JE (2016) System engineering management, 1st edn. Wiley

Booch G, Rumbaugh J, Jacobson I (1999) The unified modeling language user guide. Addison-Wesley

Borgatti SP (2005) Centrality and network flow. Social Netw 27(1):55–71

Braha D (2016) The complexity of design networks: structure and dynamics. Experimental design research. Springer, Cham, pp 129–151

Braha D, Bar-Yam Y (2004a) Information flow structure in large-scale product development organizational networks. J Inf Technol 19(4):244–253

Braha D, Bar-Yam Y (2004b) Topology of large-scale engineering problem-solving networks. Phys Rev E 69(1):016113

Brandes U (2001) A faster algorithm for betweenness centrality. J Math Sociol 25(2):163–177

Browning TR, Fricke E, Negele H (2006) Key concepts in modeling product development processes. Syst Eng 9(2):104–128

Buchanan WJ (2004) The handbook of data communications and networks. Springer. https://doi.org/10.1007/978-1-4020-7870-5

Cadini F, Zio E, Petrescu C-A (2009) Using centrality measures to rank the importance of the components of a complex network infrastructure. In: Setola R, Geretshuber S (eds) Critical information infrastructure security. Springer, Berlin, pp 155–167. https://doi.org/10.1007/978-3-642-03552-4_14

Cancela H, Petingi L (2004) Reliability of communication networks with delay constraints: Computational complexity and complete topologies. Int J Math Math Sci 2004(29):1551–1562. https://doi.org/10.1155/S016117120430623X

Chaturvedi SK (2016) Network reliability: measures and evaluation. John Wiley and Sons, Cham

Chen Z, Dehmer M, Emmert-Streib F, Shi Y (2018). In: Chen Z, Dehmer M, Emmert-Streib F, Shi Y (eds) Modern and interdisciplinary problems in network science: a translational research perspective, 1st edn. CRC Press. https://doi.org/10.1201/9781351237307

Clarkson PJ, Simons C, Eckert C (2004) Predicting change propagation in complex design. J Mech Des 126(5):788–797. https://doi.org/10.1115/1.1765117

Conchir D (2010) Overview of the PMBOK guide: Short cuts for PMP certification. Springer Publishing Company, Incorporated

Danilovic M, Browning TR (2007) Managing complex product development projects with design structure matrices and domain mapping matrices. Int J Project Manag 25(3):300–314

Dickerson C, Mavris DN (2016) Architecture and principles of systems engineering. CRC Press

Diestel R (2000) Graph theory, 2nd edn. Springer

Dissaux P, Amine MF, Michel P, Vernadat F (2005) Architecture description languages: IFIP TC-2 workshop on architecture description languages (WADL), world computer congress, aug. 22-27, 2004, Toulouse, France (Vol. 176). Springer Science and Business Media

Dorofee AJ, Walker JA, Alberts CJ, Higuera RP, Murphy RL (1996) Continuous risk management guidebook. Carnegie-Mellon Univ Pittsburgh, Cham

Engel A, Reich Y (2015) Advancing architecture options theory: Six industrial case studies. Syst Eng 18(4):396–414

Eppinger SD, Browning TR (2012) Design structure matrix methods and applications. MIT Press

Eppinger S, Ulrich K (2015) Product design and development. McGraw-Hill Higher Education

Eppinger SD, Salminen V (2001) Patterns of product development interactions. Presented at the International Conference on Engineering Design, Glasgow, UK, 21–23 August 2001

Erdos P, Rényi A (1960) On the evolution of random graphs. Publ Math Inst Hung Acad Sci 5(1):17–60

Freeman LC (1977) A set of measures of centrality based on betweenness. Sociometry 40(1):35–41. https://doi.org/10.2307/3033543

Genta G, Morello L, Cavallino F, Filtri L (2014) The motor car: Past, present and future. Springer Science and Business Media

Gross JL, Yellen J (2004) Handbook of graph theory. CRC Press

Harman T, Dabney J (2001) Mastering simulink 4. Prentice Hall, Englewood Cliffs, NJ

Haskins C, Forsberg K, Krueger M, Walden D, Hamelin D (2006) Syst Eng Handb 9:13–16

Hatala J, George Lutta J (2009) Managing information sharing within an organizational setting: a social network perspective. Perform Improv Q 21(4):5–33

Hillson D (2003) Effective opportunity management for projects. CRC Press. https://doi.org/10.1201/9780203913246

Hillson D (2014) How to manage the risks you didn't know you were taking. In: Phoenix AZ (ed) Paper presented at PMI® Global Congress 2014—North America, Project Management Institute: Newtown Square, PA, USA

Kapurch SJ (2010) NASA systems engineering handbook. Diane Publishing

Kerzner H (2017) Project management: A systems approach to planning, scheduling, and controlling, 20th edn. Wiley

Kivelä M, Arenas A, Barthelemy M, Gleeson JP, Moreno Y, Porter MA (2014) Multilayer networks. J Complex Netw 2(3):203–271

Kleinsmann M, Buijs J, Valkenburg R (2010) Understanding the complexity of knowledge integration in collaborative new product development teams: a case study. J Eng Tech Manage 27(1–2):20–32

König C, Kreimeyer M, Braun T (2008) Multiple-domain matrices as a framework for systematic process analysis. In: DSM 2008: Proceedings of the 10th International DSM Conference, Stockholm, Sweden

Kordova S, Katz E, Frank M (2019) Managing development projects—the partnership between project managers and systems engineers. Syst Eng 22(3):227–242

Kurtoglu T, Tumer IY (2008) A graph-based fault identification and propagation framework for functional design of complex systems. J Mech Des. https://doi.org/10.1115/1.2885181

LaMeres BJ (2019) Introduction to logic circuits and logic design with VHDL. Springer

Lano R (1977) The N2 chart. TRW Software Series

Lester A (2014) Project management, planning and control: Managing engineering, construction and manufacturing projects to PMI, APM and BSI standards, 6th edn. Elsevier

Li J, Zhu X, Lee C-F, Wu D, Feng J, Shi Y (2015) On the aggregation of credit, market and operational risks. Rev Quant Financ Acc 44(1):161–189

Locatelli G, Mancini M, Romano E (2017) Project manager and systems engineer: a literature rich reflection on roles and responsibilities. Int J Project Organ Manag 9(3):195–216

Marcus D (2008) Graph theory: a problem oriented approach. Maa

Maurer M, Lindemann U (2008) The application of the multiple-domain matrix: considering multiple domains and dependency types in complex product design. In: IEEE International Conference on Systems, Man and Cybernetics, pp 2487–2493

Milosevic DZ (2003) Project management toolbox: tools and techniques for the practicing project manager. John Wiley and Sons

Newman M (2010) Networks: an introduction. Oxford University Press

Nilsson J, Riedel S (2011) Electric circuits, 9th edn. Prentice Hall

Pahl G, Beitz W (1996). In: Wallace K (ed) Engineering design–a systematic approach. Springer-Verlag

Park K, Kremer GEO (2019) An investigation on the network topology of an evolving product family structure and its robustness and complexity. Res Eng Des 30(3):381–404

Patil H, Sirsikar S, Gholap N (2017) Product design and development: phases and approach. Int J Eng Res. https://doi.org/10.17577/IJERTV6IS070136

Pimmler TU, Eppinger SD (1994) Integration analysis of product decompositions. In: Proceedings of the ASME 6th International Conference on Design Theory and Methodology, Minneapolis, MN, 1994

Publishing DK (2011) Car: the definitive visual history of the automobile, 1st edn. Publishing DK

Rausand M, Hoyland A (2003) System reliability theory: Models, statistical methods, and applications, 396th edn. John Wiley and Sons, Cham

Raz T, Shenhar AJ, Dvir D (2002) Risk management, project success, and technological uncertainty. Randd Manag 32(2):101–109

Rechtin E, Maier MW (2010) The art of systems architecting. CRC Press

Reich Y, Subrahmanian E (2020) The PSI framework and theory of design. IEEE Trans Eng Manag 69:1037–1049

Renyi E (1959) On random graph. Publ Math 6:290–297

Richards M (2015) Software architecture patterns. O'Reilly Media, Sebastopol

Sage AP, Rouse WB (2014) Handbook of systems engineering and management. John Wiley and Sons

Sered Y, Reich Y (2003) Standardization and modularization driven by minimizing overall process effort. Int Des Eng Techn Conf 37017:449–458

Simon P (ed) (1997). APM Group Ltd

Summers JD, Shah JJ (2010) Mechanical engineering design complexity metrics: Size, coupling, and solvability. J Mech Des. https://doi.org/10.1115/1.4000759

Van den Brink S, Kleijn R, Sprecher B, Tukker A (2020) Identifying supply risks by mapping the cobalt supply chain. Resour Conserv Recycl 156:104743

Wasserman S, Faust K, Urbana-Champaign, S. University of I. W (1994) Social network analysis: methods and applications. Cambridge University Press

Yassine A (2004) An introduction to modeling and analyzing complex product development processes using the design structure matrix (DSM) method. Urbana 51(9):1–17

Yassine A, Braha D (2003) Complex concurrent engineering and the design structure matrix method. Concurr Eng 11(3):165–176

Yassine A, Joglekar N, Braha D, Eppinger S, Whitney D (2003) Information hiding in product development: the design churn effect. Res Eng Des 14(3):145–161