Matrix-Based Methods for Managing System Complexity

Christopher Langner[1], Tyson R. Browning[2], and Matthias Kreimeyer[1]
[1]University of Stuttgart, Germany
[2]Neeley School of Business, Texas Christian University, Fort Worth, TX, USA

Abstract

This article introduces three matrix methods - Design Structure Matrix (DSM), Domain-Mapping Matrix (DMM), and Multi-Domain Matrix (MDM) - that represent the relationships, interfaces, and dependencies among the elements comprising various systems, including those involved in the development and management of technical systems, thereby making the design of the structure or architecture of a system more transparent and analyzable, and improving system innovation and management. Matrix-based representations provide powerful, alternative viewpoints in system architecture frameworks (e.g., Browning, 2009).

Introduction

The increasing complexity of contemporary technical systems is challenging to manage, in part because of the emergent behaviors arising from the relationships among the systems' elements. While research provides well-established frameworks for modeling such complexity, many organizations miss opportunities to leverage powerful approaches for representing and analyzing these relationships. These challenges and opportunities include systems such as **product architectures** (e.g., modular designs, platform variants), **processes** (e.g., project/program activities, software execution), and **organizational structures** (e.g., team structures, communication networks), among many others.

Key drivers include the sheer number of **elements** (e.g., components, functions) and their **relationships** (e.g., interfaces, dependencies). A class of elements is referred to as a **domain**, while a class of relationships is referred to as a **dependency type**. Every system has a structure (architecture), which describes the arrangement of its elements and relationships to fulfill the system's purpose, and most systems can be viewed from the perspectives of multiple domains and views (e.g., Browning, 2009).

Matrix-based methods offer a structured way to model and view these dependencies and the patterns they create within and across domains. This article introduces three core techniques - **Design Structure Matrix (DSM)**, **Domain Mapping Matrix (DMM)**, and **Multi-Domain Matrix (MDM)** – and demonstrates their role in managing complexity in systems engineering.

Design Structure Matrix (DSM)

The **Design Structure Matrix (DSM)** is an $n \times n$ (square), domain-specific matrix method for analyzing and optimizing systems by explicitly modeling dependencies among n homogeneous elements. Originating from process engineering (Steward, 1967, 1981) and later generalized (along with N^2 diagrams (Lano, 1979)) for various system architectures (Eppinger & Browning, 2012), the DSM encodes interactions in a compact format: rows and columns represent system domain elements (e.g., activities,

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

components), while off-diagonal cells indicate directional dependencies. Diagonal cells are usually ignored, as they represent self-interactions. A binary DSM shows merely the presence or absence of each possible dependency, whereas more sophisticated DSMs may employ a variety of colors and symbols to convey additional information about the elements and/or their dependencies (e.g., type, strength, etc.). DSMs are differentiated into static and temporal DSMs. Static DSMs are typically used for describing component-based (product architectures) or people-based (organizational architecture) systems, whereas a temporal DSM represents elements related in a time dimension (e.g., activities in a process). Although there are many ways to represent systems, the DSM's power and efficacy lies in its ability to visualize and quantify structural complexity and thereby spark innovation and better management.

Key DSM analysis techniques, as illustrated in Figure 1, include:

- **Clustering** groups elements based on dependency density, enabling modular decomposition. Highly interdependent elements form clusters with sparse external links, a cornerstone of modular product architectures. Sensitivity to relationship types (e.g., prioritizing functional over spatial dependencies) is critical for aligning clusters with system goals (e.g., design for maintainability vs. functionality).
- **Sequencing (Partitioning)** reorders elements to shift dependencies to one side of the diagonal (above or below depending on the convention being used), typically isolating and minimizing feedback loops (cyclicality, potential iterations) in a temporal DSM. An ideal sequence eliminates all feedback marks, although practical systems often retain clusters of closely coupled elements near the diagonal. Sequencing reduces unnecessary rework in processes or simplifies design hierarchies.

Another advantage is the ability to represent different levels of detail in a system analysis within a single model. This makes it very easy to expand or collapse the matrix for individual subsystems, thereby enabling both a quick overview of the system at the top-level subsystem level and a detailed analysis at the lowest element level within a single model view.

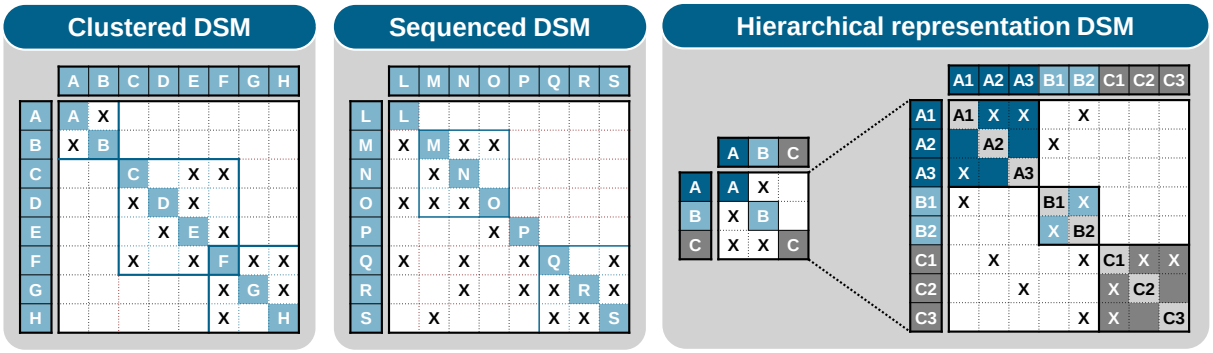


Figure 1: Generic representation of a clustered, sequenced, and hierarchical DSM

Practical DSM applications range from process optimization (reducing feedback loops causing iterations and rework) to product architecture design (modularity) and organizational restructuring (team coordination), across various industry sectors, such as automotive, aerospace, software, and construction (Browning, 2016).

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

For example, McCord and Eppinger (Eppinger & Browning, 2012) applied a DSM to General Motors Powertrain Division's engine-development program, mapping the interactions among 22 product development teams (PDTs) that were grouped into four "system teams." The DSM displayed in Figure 2 revealed numerous daily and weekly information exchanges both within and across these teams (A-V), the latter challenging their efficacy. Because many critical links spanned the prescribed boundaries, conventional integrative mechanisms (e.g., scheduled meetings, liaison roles, or distribution lists based on the original team structure) were unlikely to achieve timely coordination. A clustering analysis of the DSM, shown in Figure 3, reorganized the team structure into five teams, allowing PDTs with high interaction frequency to be co-located in the same team, even if this meant overlapping membership. The new design also featured a dedicated integration team that interacted with every other team, enabling more effective placement of integrative mechanisms reflecting the true communication needs of the program.

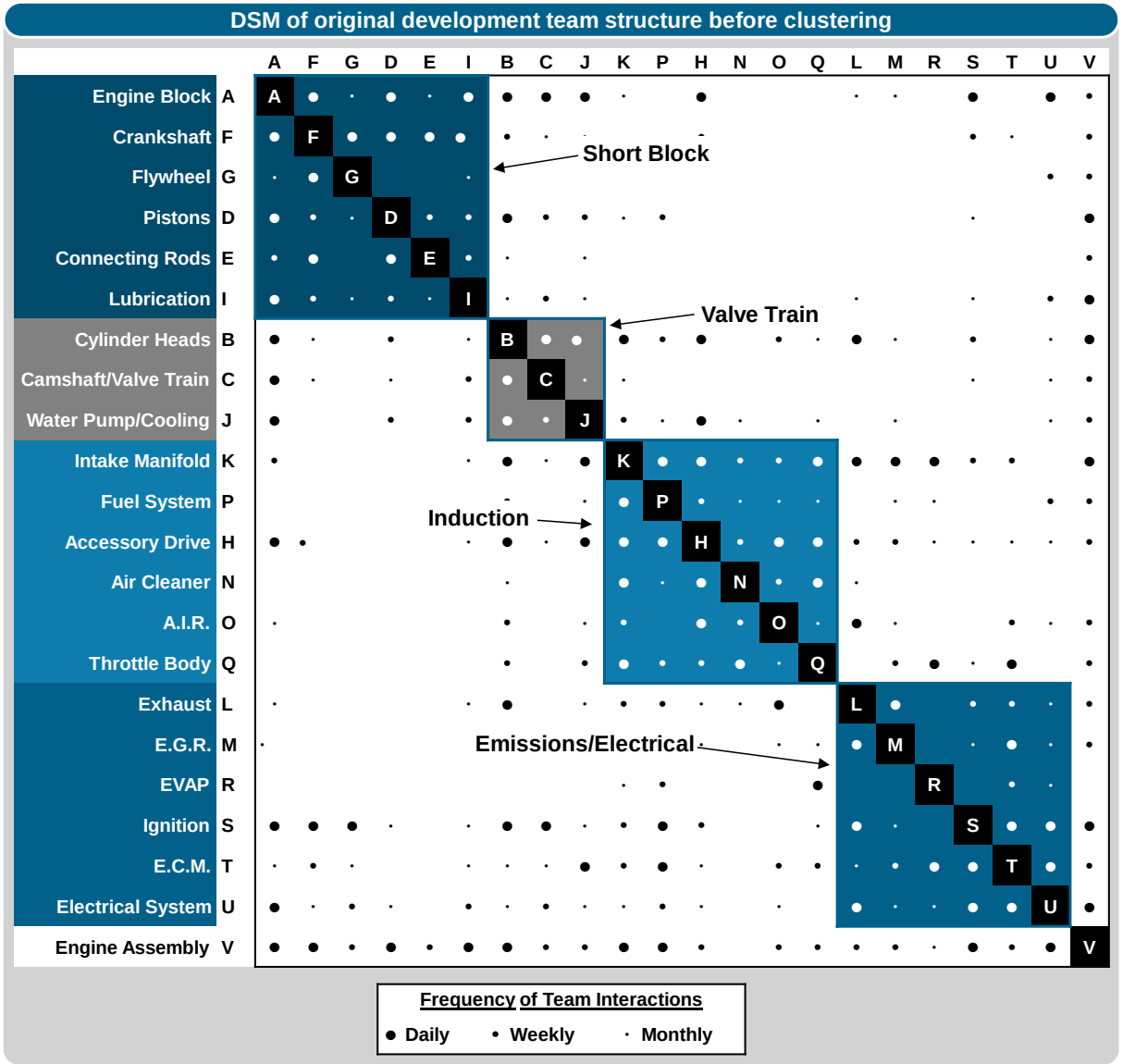


Figure 2: Original team decomposition before DSM-based analysis

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

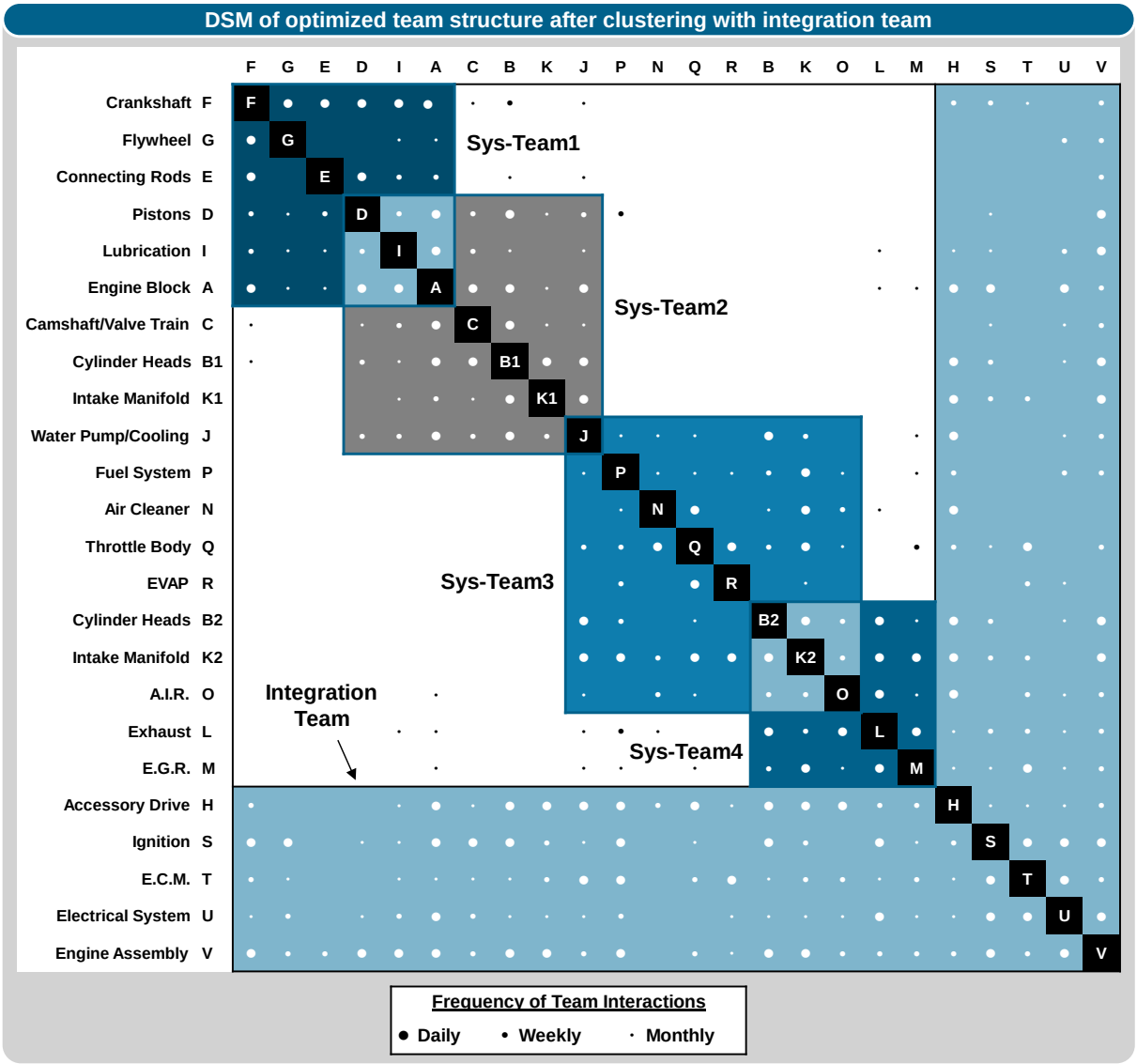


Figure 3: Clustering-based optimized team decomposition with "integration team"

Domain-Mapping Matrix (DMM)

While the DSM excels at modeling intra-domain dependencies (and can even be extended to show connections to another domain via a "1.5-domain DSM" (Eppinger & Browning, 2012)), systems engineering often requires analyzing relationships *between* distinct domains, such as functions and components, requirements and test cases, or teams and tasks. The **Domain Mapping Matrix (DMM)** addresses this need by mapping *n* elements of one domain (rows) to *m* elements of another (columns), enabling **cross-domain integration** and traceability (Danilovic & Browning, 2007). Unlike the always-square DSM, the DMM is often rectangular and directional: a cell at row *i* and column *j* signifies a dependency of *i* on *j*, while the transposed DMM reverses the interpretation. (If relationships between two domains are directional, then $DMM_{nm} \neq DMM_{mn}$.) Figure 4 shows a DMM example of mapping customer requirements to product specifications, such as in a Quality Function Deployment (QFD) matrix, adapted from (Danilovic & Browning, 2007). Based on clustering, the DMM analysis reveals how a specific set of customer needs is met by a corresponding set of product features – or, conversely, how a particular set of product characteristics satisfies

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

customer requirements. In traditional QFD analysis, this level of interdependency typically occurs on an individual element basis, rather than at a set level. Figure 4 highlights four major and one minor cluster, as well as an area with no interdependencies between customer requirements and product specifications. While the clusters indicate areas that require intense coordination and integration between the two domains, the latter area reveals a discrepancy between two product specifications and the identified customer requirements, suggesting that vital information may have been lost during the product design phase or that specifications were added without corresponding customer needs. In both cases, an inconsistency exists between the requirements and specifications domain, which is identified by the DMM analysis.

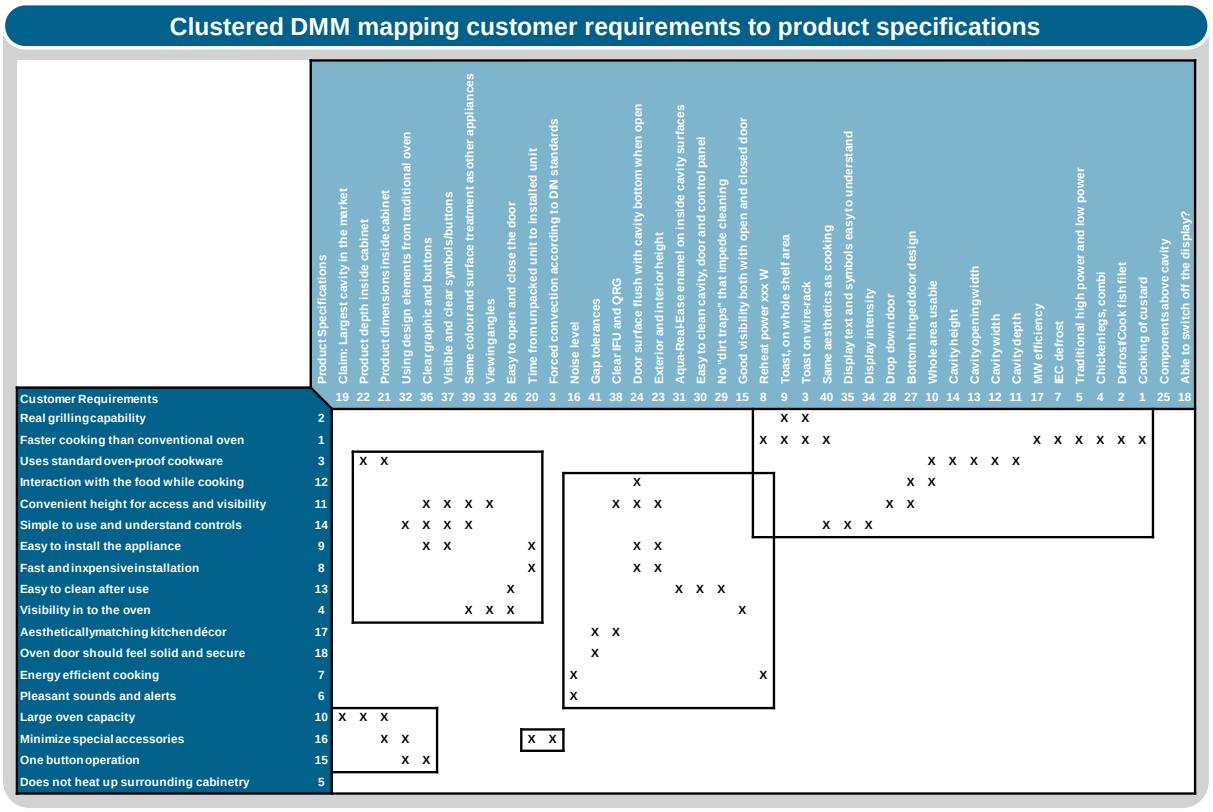


Figure 4: Example of a DMM mapping customer requirements to product specifications

The DMM's primary utility lies in **bridging silos** and integrating between systems or across a system's subdomains. Key applications include:

- **Requirements Allocation:** Mapping stakeholder requirements (one domain) to design parameters, product components, or verification methods (other domains) ensures traceability and completeness. For example, a functional requirement ("vehicle must brake within 50 m") can be linked to specific sensor specifications, or test procedures.
- **Interface Management:** DMMs visualize cross-domain interfaces, such as the relationship between **software modules** and **hardware components**, or **process steps** and **organizational teams**. This is critical for identifying gaps, redundancies, or misalignments (e.g., a task assigned to a team lacking the necessary skills, or a task for which no team has been assigned).

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

- **Modularity and Decomposition:** By comparing DMMs and DSMs, engineers can evaluate modularity trade-offs. For instance, a DMM linking **customer features** to **product subsystems** can expose "tangles," where a single feature spans multiple subsystems, suggesting opportunities for redesign.
- **Change Impact Analysis:** When a change occurs in one domain (e.g., a modified component), the DMM propagates the impact to dependent elements in other domains (e.g., affected functions, requirements, or teams), supporting risk assessment and change management.

Danilovic and Browning (2007) highlighted the DMM's role in **closing the gap between abstraction layers**, a challenge central to holistic system design. For example, mapping **logical functions** (e.g., "generate thrust") to **physical components** (e.g., engines, propellants) bridges conceptual and realized architectures. However, DMMs require careful attention to **cardinality** (one-to-one, one-to-many, or many-to-many relationships) and **relation semantics** (e.g., "requires," "controls," or "verifies"). Weighting, symbolizing, or color-coding cells can further enhance interpretability for complex systems. Many formal DMMs have been primarily analyzed through clustering. DMMs are generalizations of common methods and tools using rectangular matrices to map between domains, such as QFD and Responsible-Accountable-Consult-Inform (RACI) charts, which further suggests their potential for extendibility and customization in a variety of industries and applications (Browning, 2016).

In practice, DMMs are often used **in tandem with DSMs**, e.g., a DSM modeling dependencies among components with a DMM linking those components to their supporting functions. This hybrid approach, formalized in the **Multi-Domain Matrix (MDM)** framework, enables comprehensive system analysis, aligning with the principles of **integrated, multi-perspective engineering**.

Multi-Domain Matrix (MDM)

The **Multi-Domain Matrix (MDM)** is the most comprehensive matrix-based approach for modeling and analyzing complex systems across multiple domains. As a structured composite of DSMs and DMMs, the MDM (Figure 5) arranges these matrices as blocks along its diagonal (intra-domain DSMs) and off-diagonal (inter-domain DMMs), potentially yielding a "DSM of DSMs". This unified representation enables engineers to simultaneously capture and evaluate dependencies *within* domains (e.g., component-to-component interfaces) and *between* domains (e.g., functions-to-components, requirements-to-tests), addressing the limitations of isolated DSM or DMM analyses (Eppinger & Browning, 2012).

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

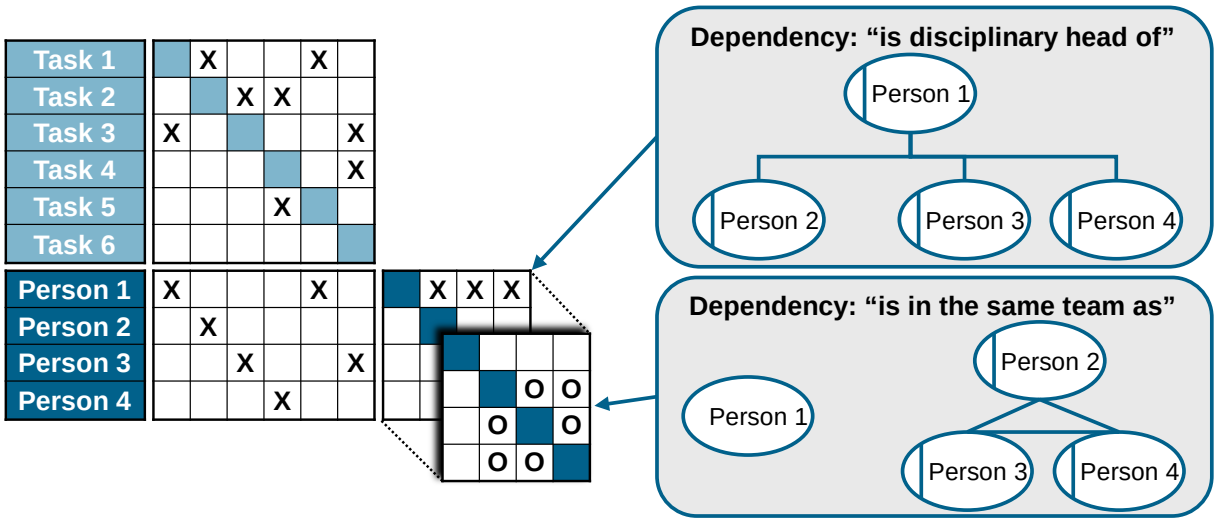


Figure 5: Generic representation of an MDM between the domains of tasks and people

The MDM’s primary advantage lies in its ability to illuminate cross-domain interactions that single-domain approaches often obscure. Practical applications include:

- **Holistic System Integration**: By consolidating physical, functional, process, and organizational perspectives (e.g., linking hardware components, software modules, and development teams and flows), MDMs reveal hidden dependencies and indirect relationships that could lead to integration failures. For example, a change in a mechanical component’s design (DSM) can be traced to affected software functions (DMM) and impacted development tasks (DSM), ensuring consistency across domains.
- **Automated Dependency Propagation**: MDMs support computational derivation of implicit relationships. For instance, if a DMM maps requirements to components, and a DSM models component interfaces, the MDM can infer *secondary dependencies*, such as how a change in one component cascades to functionally related requirements. This accelerates impact analysis and reduces manual error.
- **Modularity and Decoupling**: Engineers can identify opportunities to isolate domains by analyzing off-diagonal sparsity (e.g., minimizing cross-domain dependencies between software and hardware). High-density blocks in the MDM highlight tightly coupled areas, guiding decisions on modularization or interface standardization.
- **Lifecycle Traceability**: From conceptual design to verification, MDMs ensure traceability across evolving system representations. For example, initial functional DSMs can be progressively linked to physical DSMs and test-case matrices, enabling end-to-end validation of design decisions.
- **Organizational Alignment**: MDMs expose misalignments between technical and organizational domains (e.g., mapping teams to tasks and subsystems). Poor alignment often leads to communication bottlenecks or redundant work, which the MDM visualizes as dense cross-domain links.

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

Beyond visualization, the MDM framework enables advanced analyses through algebraic operations on its blocks. A key operation is the **matrix multiplication**, which can be used to derive new DMMs (e.g., functions-to-tests by combining functions-to-components and components-to-tests matrices) and DSMs (e.g., by multiplying a DMM by its transpose).

Industrial applications of MDMs mainly cover product and service design and process management in automotive, architecture/construction, and energy/utilities-related industries, but also aerospace and software development (Browning, 2016).

Figure 6 shows an example MDM conceptualizing the first BMW hybrid vehicle architecture (Eppinger & Browning, 2012). The dependencies among and between functions and components were investigated to understand the structure of the system compared to conventional concepts, and to further compare different hybrid architectures with each other. Comparing a series of eight MDMs (the other seven not shown here), one per architecture concept, highlighted similarities and differences between the concepts.

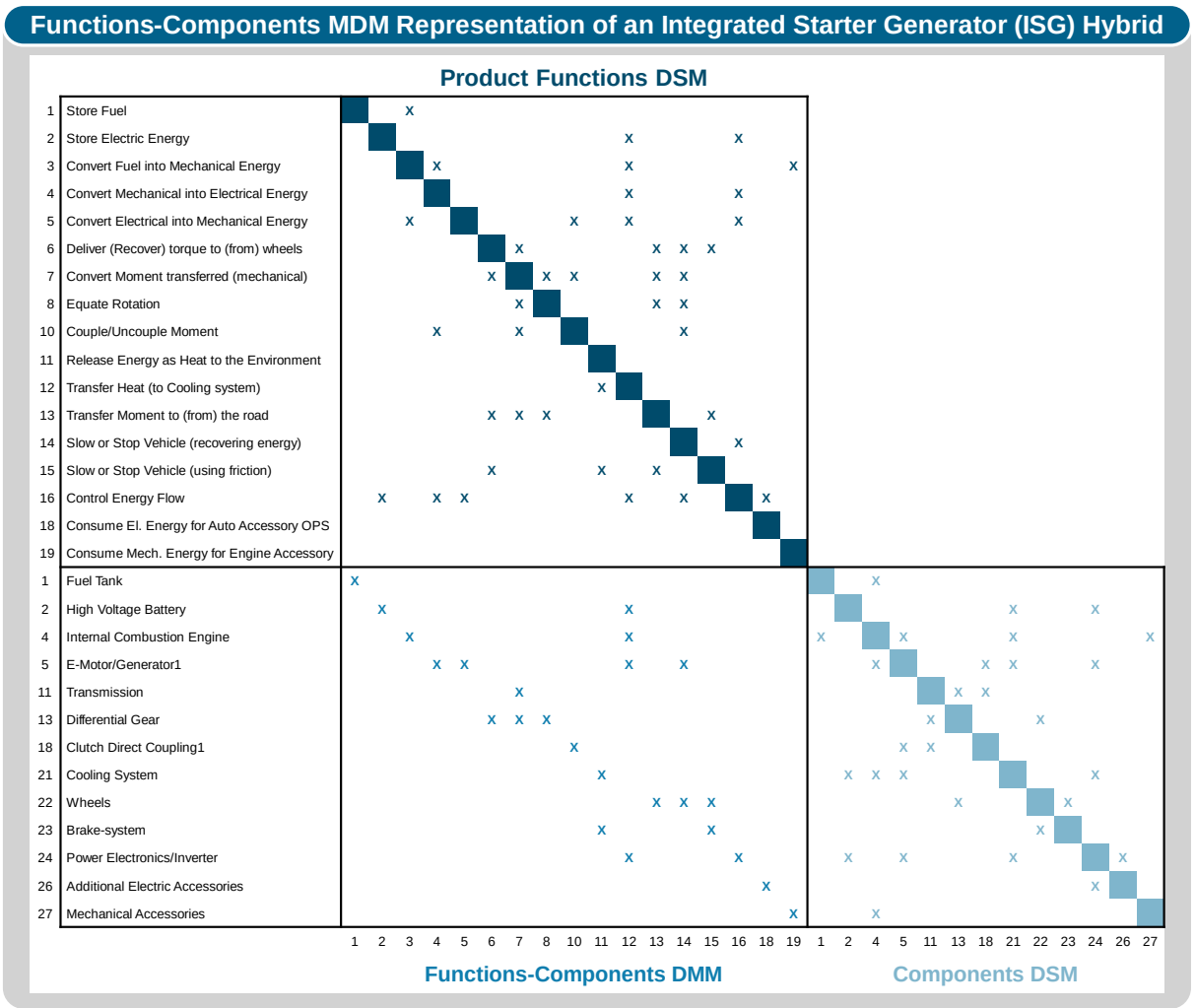


Figure 6: MDM representation of dependencies between functions and components of an integrated starter generator hybrid (ISG)

While powerful, MDMs demand rigorous data curation to avoid scalability issues. Large systems may yield unwieldy matrices, necessitating abstraction (e.g., aggregating

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

elements into subsystems) or software tools for computation. Additionally, populating an MDM requires domain expertise to define meaningful relationships and avoid spurious connections. By bridging fragmented views of system complexity, the MDM transforms compartmentalized snapshots (e.g., separate DSMs for mechanics and electronics) into a coherent, actionable model, which is critical for managing today's interconnected technical and sociotechnical systems. Future extensions, such as integrating dynamic MDMs for agile development or AI-assisted pattern detection, promise to further expand its role in systems engineering.

Recommendations for Applying Matrix-Based Methods

The described methods provide structured approaches to managing complexity in systems engineering, but their effectiveness depends on rigorous application. To maximize their value, consider the following best practices:

Scope and Granularity

- Matrix methods excel at analyzing bounded subsystems but struggle with monolithic representations of entire systems (e.g., a full vehicle with 50,000+ elements).
- Decompose complex systems hierarchically.
- Model subsystems (e.g., powertrain, chassis) separately and then integrate them into higher-level matrices (Browning, 2002).
- Focus on problem-relevant subsets to avoid unmanageable matrices while preserving meaningful insights.

Clarity in Relationship Definition

- Avoid ambiguous dependencies (e.g., "Element A depends on B").
- Capture only direct dependencies in each matrix. Then, analyze the matrix (or matrices) to infer indirect dependencies.
- Specify *how* and *why* (e.g., "geometric interface," "control signal," "shared resource") each dependency or relationship exists.
- Poorly defined relationships yield vague conclusions; precise semantics (e.g., "Component X provides torque to Component Y") sharpen analysis.
- For multi-domain models, explicitly define inter-domain linkages (e.g., "Function A *requires* Parameter B from Component C").

Validation and Sensitivity Testing

- Like any model, matrix-based models provide a foundation for analyses but should not be expected to provide "the" optimal answer. A variety of optima may emerge as different objectives and options are compared. Matrix-based models support decisions but do not make them (Eppinger & Browning, 2012).
- Matrix outputs are only as reliable as their input data. Therefore, validate relationships through expert review and cross-checks (e.g., traceability matrices for requirements).
- Highlight "critical paths", derivable from dependencies with outsized influence on system behavior, and verify their legitimacy (e.g., does a direct link between Process Steps A and C actually pass through Document B?).

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

- Missing intermediary domains (e.g., information flows between process steps) can distort results.

Tool Support and Scalability

- Leverage software tools (e.g., Excel, Matlab, or specialized platforms like Lattix) to handle large matrices and automate analyses (e.g., clustering, path tracing).
- For agile environments, consider dynamic MDMs that adapt to evolving system definitions (e.g., linking changing requirements to design parameters in real time).

Integration with Systems Thinking Principles

Align matrix methods with holistic systems thinking:

- **Traceability**: Use DMMs/MDMs to link lifecycle phases (e.g., requirements → design → verification).
- **Modularity**: Apply DSM clustering to identify cohesive modules with minimal external dependencies.
- **Risk Management**: Propagate changes via MDMs to assess impact across domains (e.g., how a component modification affects functions, tests, and teams).
- **Knowledge Management**: Document outcomes as "structural handbooks" to codify design rationale (e.g., "Why was this interface chosen?").

Contextual Awareness

- Remember that matrices abstract structural relationships without necessarily capturing behavior.
- Complement matrices with other models (e.g., SysML, simulations) to validate assumptions (e.g., does a clustered module degrade performance under load?).
- Use matrices not as standalone solutions but as part of a broader modeling toolkit.

Continuous Improvement

- Iteratively refine models as system knowledge grows. Initial DSMs for a novel engineering artifact subsystem may start coarse but evolve into detailed references for future projects.
- Treat matrices as living artifacts. Update them to reflect lessons learned and changing requirements. They provide a powerful scaffolding for knowledge management, especially about integration and the so-called “white space” between better-known domains (Eppinger & Browning, 2012).

Concluding, matrix-based methods are powerful, but, like any other method, not infallible. Their strength lies in revealing patterns and dependencies that intuition or siloed models might miss. By applying them within bounded scopes and with clear definitions, they become invaluable for navigating complexity, increasing integration, and providing alternative views leading to key insights.

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

Literature

Used literature

Browning, T.R. (2002) "Process Integration Using the Design Structure Matrix," *Systems Engineering*, 5(3): 180-193.

Browning, T.R. (2009) "The Many Views of a Process: Towards a Process Architecture Framework for Product Development Processes," *Systems Engineering*, 12(1): 69-90.

Browning, T.R. (2016) "Design Structure Matrix Extensions and Innovations: A Survey and New Opportunities," *IEEE Transactions on Engineering Management*, 63(1): 27-52.

Danilovic, M., & Browning, T.R. (2007) "Managing Complex Product Development Projects with Design Structure Matrices and Domain Mapping Matrices," *International Journal of Project Management*, 25(3): 300-314.

Eppinger, S.D., & Browning, T.R. (2012) *Design Structure Matrix Methods and Applications*, Cambridge, MA, MIT Press.

Lano, R.J. (1979) *A Technique for Software and Systems Design*, New York, NY, North-Holland.

Steward, D.V. (1967) "The Design Structure System," General Electric report 67APE6, San Jose, CA.

Steward, D.V. (1981) "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Transactions on Engineering Management,* 28(3): 71-74.

Further useful

Engel, A., Browning, T.R., & Reich, Y. (2017) "Designing Products for Adaptability: Insights from Four Industrial Cases," *Decision Sciences*, 48(5): 875-917.

Lindemann, U., Maurer, M., & Braun, T. (2009) Structural Complexity Management, Berlin, Germany, Springer.

An extensive list of DSM-related publications can be found in (Browning, 2016) and at: https://dsmweb.org/

© 2026 C. Langner, T.R. Browning, M. Kreimeyer. All rights reserved.

Copyright and Usage Rights

© 2026 Christopher Langner, Tyson Browning, Matthias Kreimeyer. All rights reserved.

This publication is made freely available for personal reading, downloading, and academic reference.

Except as permitted by applicable law, no part of this publication may be reproduced, republished, redistributed, adapted, or incorporated into other publications or materials without the prior written permission of the relevant rights holders. This includes the republication or redistribution of the complete document and the reuse of individual figures, tables, or substantial portions of the text.

Proper citation and acknowledgment of the authors are required when referring to this publication. Any use of the work remains subject to applicable statutory exceptions and limitations under copyright law.

No general license for further use or redistribution is granted by making this publication freely accessible.

All rights in the work remain with their respective rights holders, subject to any prior agreements, third-party rights, and applicable law.

Cite as: “Langner, C., Browning, T.R., & Kreimeyer, M. (2026). *Matrix-Based Methods for Managing System Complexity*. https://doi.org/10.35199/report.DSMbasics.”