International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 11(2019) pp. 1-12
https://doi.org/10.70917/ijcisim-2026-5985
Article

Designing Reusable Digital Building Blocks for Health Insurance Sales Platforms: A Metadata-Driven Architecture for Product and Quote Configuration

Susil Kumar Sahu

Salesforce Solution Architect, BlueCross BlueShield of Tennessee

Abstract: Health insurance sales applications need to deal with continuously evolving products, benefit designs, eligibility requirements, pricing rules, approval policies and regulatory requirements. The typical approaches tend to be coded into bespoke application logic, causing the duplication of functionality and an increasing maintenance burden when new products and markets enter the picture. This article suggests an architecture for building reusable digital building blocks to support health insurance sales and quoting systems. The model contains configurable components for common recurring tasks such as employer information collection, census processing, benefit selection, plan configuration, quote generation, routing for approval, forms, and downstream integration. The user experience is separated from product and business rules by a metadata-driven configuration layer, enabling the introduction of product and validation changes without significant redevelopment of the user experience. Reusable integration interfaces are standardized services that link front-end applications to policy, enrollment, pricing, document-generation and administrative systems. The framework suggests that enterprise insurance applications will continue to transition from monolithic enterprise process implementations to libraries of reusable domain capabilities, which will deliver greater consistency in enterprise applications across products and markets, will lessen the need for enterprise application duplication, and will enable composable enterprise applications. Based on a synthesis of 29 sources published 1972 to 2018, break-even occurs at six products, with a 37.2% reduction of cumulative effort at 12 products, and a 69.1% reduction in mean change lead time.
Keywords: health insurance, digital building blocks, metadata-driven architecture, product configuration, quote generation, software reuse, feature models, model-driven engineering, composable enterprise, insurance platforms

1. Introduction

Digitalisation is transforming the distribution of health insurance, as it is for all insurance products, and influences all aspects of the insurance value chain, from product development, marketing to underwriting, pricing and claims. Employer-sponsored group coverage is especially vulnerable to this shift as every single sale involves employer data, counting of eligible employees, benefit and plan choice, a priced quotation, and an approval decision. The logic for the above tasks is implemented as code in the screens and services used in conventional sales applications, creating new custom code for each new product or market.

InsurTech has been described with a set of transformational capabilities to change the way value is added in a digital world [2]. In sales platforms, these features are dependent on how quickly new propositions can be set up, priced, and sold. Immersing product logic into the application code introduces the risk of release cycles aligning with the capacity of the development team, and changes to benefit structures, eligibility requirements, regulatory constraints, and other engineering priorities become a clash.

Digital infrastructures as a shared, evolving, open and heterogeneous installed base with a generative characteristic that enables the emergence of new services from existing components [3] If you see this way, an


Copyright: © 2026 by the authors

insurance sales platform is not a series of separate projects, but an ongoing infrastructure development process. The presence of processing functions like census processing, or generating quotes, is required multiple times by various products, and can thus be considered as a candidate for the reusable digital building blocks which are maintained once, but assembled many times.

Digital innovation management research highlights the convergence and generativity of the digital technologies, affecting both the results and the innovation processes [4]. Within this context, 29 sources have been synthesized from 1972 to 2018, and a reference architecture for product and quote configuration has been developed based on this. Three contributions are made: A layered building-block model, a metadata layer that separates business rules from code, and an analytical model of reuse economics. The rest of the article outlines the review method, the theoretical underpinnings, the architecture, the evaluation, and implications.

2. Review Method

The design science paradigm is followed in the synthesis of purposeful artifacts which are built and tested to solve a specific organizational problem, with seven guidelines to address problem relevance, rigor and result communication [5]. In the current context, the artifact is a reusable sales capabilities architecture and the problem solved is duplicated hard-coded product logic. Since there are no deployment data from an insurance setting, evaluation takes place analytically and all numerical parameters are clearly named as assumptions.

The six activities of a design science research process model [6] were used in the review process. A total of 46 years was observed and eight themes were assigned to the sources as summarized in Table I. The concepts found at each source were analyzed for transferability to insurance sales applications in which they could be configured, and then the concepts were correlated with the architectural components presented in Section IV. In cases where the same idea was expressed in multiple ways, the most specific way was kept, and additional ways to express the idea were added together.

TABLE I. Mapping of Reviewed Literature to Concepts

Theme	Concept applied	Sources	Count
Digital transformation	Value-chain change; layered digital architecture	[1]–[4], [14]	5
Design science	Artifact evaluation; research process	[5], [6], [29]	3
Modularity and reuse	Information hiding; reuse economics	[7]–[10], [25]	5
Variability	Feature models; configuration validity	[11]–[13]	3
Model-driven engineering	Metadata, languages, transformations	[17]–[20], [27]	5
Services	Service contracts; microservices	[15], [16], [21], [28]	4
Process management	Process models; workflow patterns	[22]–[24]	3
Software evolution	Continuing change of systems	[26]	1
Total			**29**

3. Theoretical Foundations

A. Modularity and Reuse

Modular decomposition was found to be most effective when the modules are based on design decisions that are likely to change, and each design decision is hidden behind a stable interface [7]. Structures of benefits, rating factors, eligibility criteria, and approval limits are just such changing decisions in health insurance sales. This means

that the definition of each reusable building block is determined by the concern that it encapsulates, and not by the screen that it's displayed on.

Flexibility in product design has been linked to modularity with the idea that modular design means that the interface between the components is standardised, so that variants are produced by recombination and not redesign [8]. This is what interfaces (between blocks) do in a sales platform, for example, exchanging a validated census or a selected plan set. Loose coupling at these points allows for individual blocks to be modified, replaced or reused without affecting the rest of the customer journey.

A software reuse definition according to the process of abstraction, selection, specialization and integration of the software reusables [9]. How much a platform reuses can be expressed as a reuse ratio: *Nr*/*Nt* as defined in (1) where *Nr* is the number of capabilities of a product journey that is supplied by shared building blocks, and *Nt* is the total number of capabilities required for that product journey.

$$\rho = \frac{N_r}{N_t} \quad (1)$$

According to the reviews of the reuse research, reuse is still multidimensional problem, technical, management, economic and legal, and domain specific and systematic approaches are considered as the most promising direction [10]. The building blocks offered here, for this reason, are limited to the domain of group health insurance sales; a system of ownership, versioning and retirement of blocks is considered necessary and is seen as a prerequisite to the build blocks.

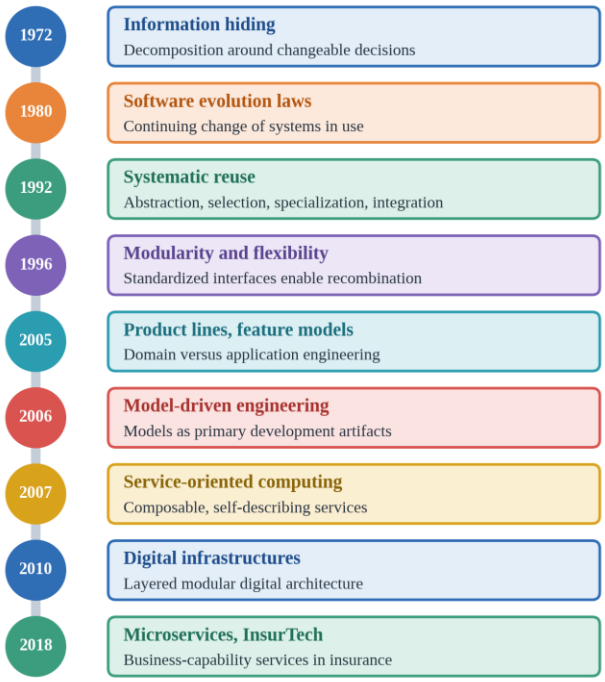


Fig. 1. Evolution of the Underlying Concepts, from Modular Decomposition to Digital Infrastructures

B. Variability Management

A software product line engineering approach is based on the partitioning of domain engineering, which establishes the reusable assets and variability of a software product line, and application engineering, which generates individual products from the assets of the software product line [11]. For health insurance sales, the separation is applied at the domain level where the variability of the product or the market is captured once and at the product launch level, where each medical plan (small group) or dental plan offering is configured with the points of variability.

Equivalently, feature models can be represented as grammars, and also as propositional formulas for which logical reasoning can be used to verify configurations validity [12]. Suppose that M is a feature model that contains K constraints, denoted by ψk, and c is a candidate configuration of plans, riders and options. The configuration is valid if (2) holds true, i.e., when all the constraints are satisfied at the same time.

$$c \in L(M) \Leftrightarrow \Lambda_{k=1}^{K} \psi_k(c) = 1 \quad (2)$$

Over 53 main studies, automated analysis of feature models has been studied, resulting in 30 main operations that can be performed on feature models, such as checking their validity, counting the number of valid feature products, and searching for dead features [13]. These operations are applicable to the case of plan catalogues where incompatible riders or inaccessible options need to be identified prior to the publication of a product. Validation is thus performed at authorisation time, not at point of sale.

4. Metadata-Driven Architecture

A. Layered Reference Architecture

Digital technology is often described as having a layered modular architecture, with technology devices, networks, services, and content being loosely coupled and independent of each other [14]. A similar layering is suggested for health insurance sales in Fig. 2. Experience channels are the first layer, followed by a library of reusable building blocks, a metadata and rule layer, an integration services that are standardized, and then a policy, enrollment, pricing, document-generation, and administrative systems layer. Dependencies are only allowed between adjacent layers and can't go directly into core systems through experience channels.

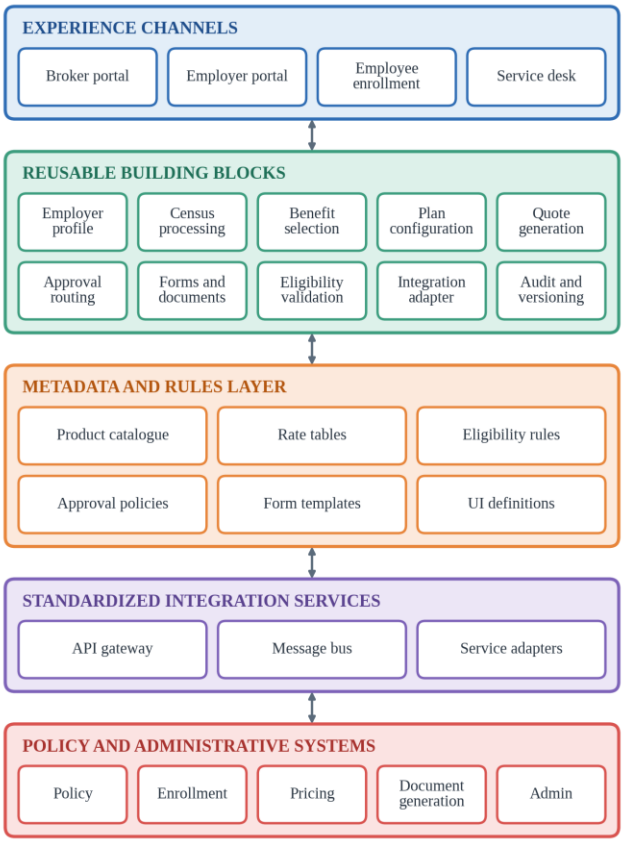


Fig. 2. Layered Reference Architecture for Health Insurance Sales Platforms

C. Reusable Building Blocks

Self-describing services that are platform-independent and can be discovered and composed into distributed applications are the tenets of service-oriented computing [15]. Ten building blocks are characterized in this way, as shown in Table II. Each block publishes a service contract, takes in a set of metadata that dictate how it operates, and interacts with downstream systems through the integration layer. Medical, dental, vision, and life products share employer profile and census blocks, as is, rather than being modified.

TABLE II. Catalogue of Reusable Building Blocks

Building block	Primary responsibility	Metadata consumed
Employer profile	Capture group size, industry, location	Field definitions
Census processing	Import, cleanse, classify employees	Validation rules; tiers
Benefit selection	Present eligible benefit categories	Product catalogue
Plan configuration	Select plans, riders, contributions	Feature model
Quote generation	Rate census, assemble quotation	Rate tables; loadings
Approval routing	Route deviations to authority levels	Approval policies
Forms and documents	Assemble forms and proposals	Form templates
Eligibility validation	Evaluate rules across the journey	Eligibility rules
Integration adapter	Exchange data with core systems	Service mappings
Audit and versioning	Record decisions and rule versions	Effective-dated records

Microservices are an up-to-date improvement on the service orientation [16] approach by having small services, independently deployable, grouped around business capabilities. Granularity of blocks is thus aligned with the business capability rather than technical layers. Not all blocks need to be deployed as individual services: if a block changes infrequently, it can be packaged with other blocks that change more often; if a block changes often, such as rating and approval blocks, it can be released separately.

D. Metadata and Configuration Layer

Model driven engineering (MDE) increases the level of abstraction by considering models as main development artifacts from which implementation artifacts are generated or interpreted [17]. As stated above, in the suggested architecture, product definition, rate tables, eligibility rules, approval policies, form templates, and interface definitions are all represented as versioned metadata, which is then interpreted during runtime by the building blocks, as shown in Table III. There is a review path for changes that is specific to each metadata type, and it is determined by the metadata type owner.

TABLE III. Metadata Artifact Types and Ownership

Artifact	Content	Typical owner
Product catalogue	Plans, riders, feature constraints	Product management
Rate tables	Tier rates, loading factors	Pricing
Eligibility rules	Class, hours, waiting-period conditions	Underwriting
Approval policies	Authority thresholds, routing chains	Sales operations
Form templates	Field mappings, market-specific forms	Compliance
Interface definitions	Screen layout, labels, field order	Experience design

It has been demonstrated that full code generation can be achieved in constrained domains using domain-specific model languages, where the modeling language is based on the vocabulary of the problem domain [18].

Product specialists can then define and keep plans and rules as a notation reflecting business jargon, including coverage tiers, contribution rules, and waiting periods, and let platform teams handle implementation details.

There are two kinds of decision patterns that can be used to determine whether to create a domain-specific language are identified: A problem domain that can be formalized and specifications that change often are appropriate for the development of a domain-specific language [19]. Premium calculation meets both requirements. If the census is divided into a set of J rating tiers, with *nj* members rated at *rj*, and a loading factor λ for taxes, fees or group adjustments, then the total quoted premium is calculated from (3). All tier rates and loading factor are provided in the metadata.

$$P = (1 + \lambda) \sum_{j=1}^{J} \quad n_j r_j \qquad (3)$$

There are several models of model transformation approaches: the form of rules, directionality and the nature of the application of the rules [20]. In current architecture, the metadata models are converted to executable forms such as interface definition, validation rules, rating tables and document mappings as shown in Fig. 3. This technique is compared with code-embedded and parameter-driven implementations in Table IV, and it can be seen that only the metadata-driven approach has business-owned change and systematic validation.

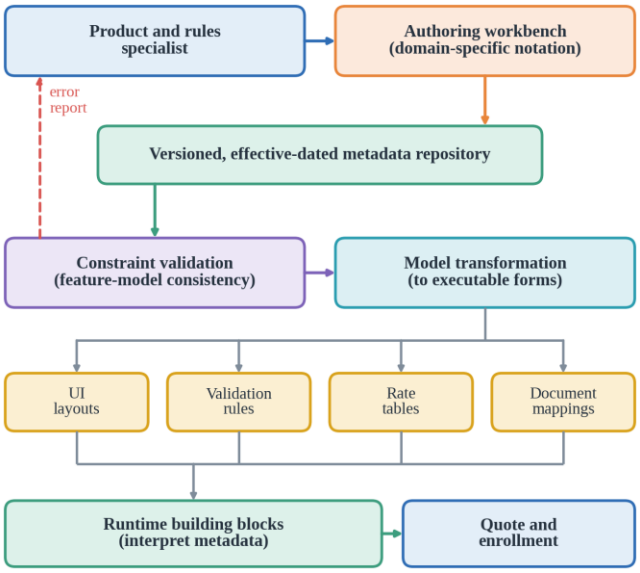


Fig. 3. Metadata-Driven Configuration Pipeline from Authoring to Runtime

TABLE IV. Comparison of Implementation Approaches

Criterion	Code-embedded	Parameter-driven	Metadata-driven blocks
Logic location	Application code	Code plus settings	Versioned metadata
Change path	Release cycle	Release or setting	Metadata publication
Variant handling	Branching code	Limited switches	Feature-model binding
Reuse across products	Low	Moderate	High
Validation	Manual testing	Basic checks	Automated constraint checks
Traceability	Source history	Partial	Effective-dated versions

E. Integration and Process Orchestration

Progress toward service-based architectures has been claimed as a gradual process, where services are progressively separated and extracted from the existing system, while it is still running [21]. For legacy sales applications, the same approach can be applied: the census and quote capabilities—the most commonly duplicated—are converted first with a standard interface, and downstream systems are accessed in the meantime via adapters during the switch. Before retiring the original code, each step is tested against the legacy code to assure that the code produces the same effect.

Business process management has a lifecycle for the identification, modelling, analysis, redesign and monitoring of business processes [22]. The quote to enrollment process is then modelled as a public process with tasks that are explicitly associated with building blocks, thus decoupling the order of the activities from the implementation of each activity. Revising the sequence of steps to capture, census, and select the plan only necessitates a revised model, not the redevelopment of screens.

Process engines run process models and evaluate branching conditions that are stored in data and not as program code [23]. Approval routing is an example of a representative case. Let dq be the discount/Deviation requested for a quote q, and let $\theta 0 < \theta 1 < \ldots < \theta K$ be the authority thresholds retrieved from metadata. The level of responsibility for approving is established as in (4) ($A0$ denotes automatic approval, AK+1 denotes escalation to highest authority).

$$R(q) = \{A_0,\ if\ d_q \le \theta_0\ A_k,\ if\ \theta_{k-1} < d_q \le \theta_k,\ k = 1, \ldots, K\ A_{K+1},\ if\ d_q > \theta_K \qquad (4)$$

A set of 20 workflow patterns have been catalogued [24] that include control-flow patterns like sequence, exclusive choice, parallel split, and synchronization. The combination of these four patterns is seen in the approval journey in Fig. 4: The threshold test is an exclusive choice, underwriting and compliance reviews are a parallel split with synchronization, while the rest of the steps are sequential. The same routing block can be used for products having different approval chains, since the pattern configuration is stored as metadata.

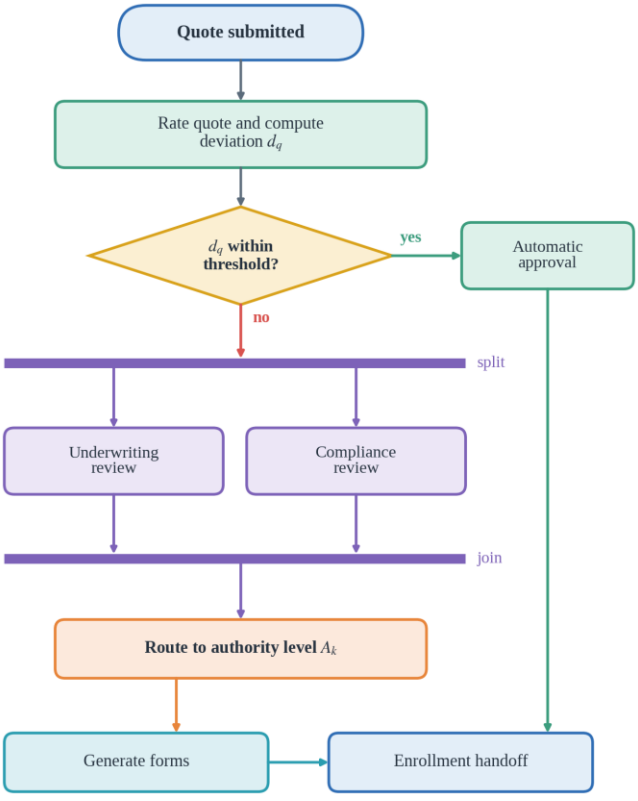


Fig. 4. Approval Routing Workflow Combining Exclusive Choice, Parallel Split, and Synchronization

5. Analytical Evaluation

Previous industrial research on the reuse examined quality, productivity and economic implications; and these have been found to be improved, with the caveat that there must be an initial investment in reusable assets in order to achieve these benefits [25]. The effort model is then formulated as follows: an effort number n for the number of products delivered, a product core effort *C*core for the effort required for the implementation of the core journey logic that is common to all products, a product variant effort *C*var for the effort required for the implementation of the product-specific variant of the common journey logic, a fixed investment effort *Cf* for the construction of the building blocks and the metadata layer, and a residual variant effort *C*res for the effort required for the implementation of the product variant of the common journey logic not covered by the reuse ratio (1). Together with ρ = 0.75, it is assumed that 24, 6, 130, and 2 person-weeks, respectively, are taken. The cumulative effort for both methods will follow (5) and (6), and the break-even quantity of products will follow (7).

$$E_b(n) = n(C_{core} + C_{var}) \qquad (5)$$

$$E_m(n) = C_f + n[(1-\rho)C_{core} + C_{res}] \qquad (6)$$

$$n^* = \frac{C_f}{\rho C_{core} + C_{var} - C_{res}} \qquad (7)$$

Under the assumed parameters, break-even is reached at $n^* = 5.9$, that is, at the sixth product, as shown in Table V and Fig. 5. At 12 products, cumulative effort falls from 360 to 226 person-weeks (37.2%), and at 24 products from 720 to 322 person-weeks (55.3%). Sensitivity analysis in Fig. 6 shows that break-even ranges from 13.0 products at ρ = 0.25 to 5.1 products at ρ = 0.90 when *Cf* = 130. Because software systems in use must continuously change [26], change lead time is also decisive. Table VI and Fig. 7 compare six change types, for which mean lead time falls from 7.8 to 2.4 weeks; the aggregate relative reduction, defined in (8), is 69.1%, where *Tb* and *Tm* denote the total lead times of code-embedded and metadata-driven changes, respectively.

$$\Delta = \frac{T_b - T_m}{T_b} \times 100\% \qquad (8)$$

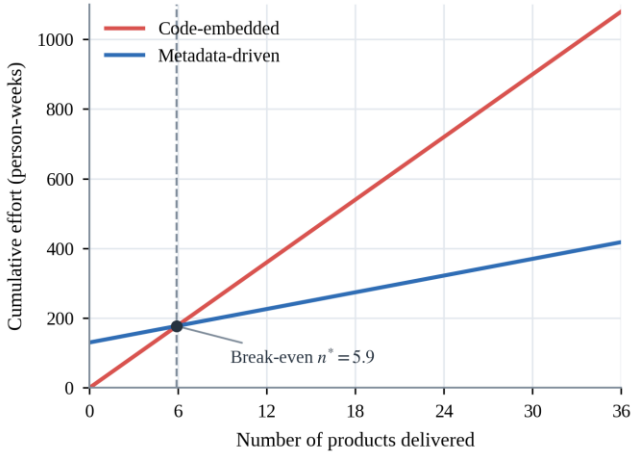


Fig. 5. Cumulative Effort of Code-Embedded and Metadata-Driven Implementations

TABLE V. Cumulative Effort under Assumed Parameters

Products, n	Code-embedded	Metadata-driven	Saving (pw)	Saving (%)
6	180	178	2	1.1
12	360	226	134	37.2

24	720	322	398	55.3
36	1080	418	662	61.3

Effort in person-weeks (pw). Assumed: *C*core = 24, *C*var = 6, *Cf* = 130, *C*res = 2, ρ = 0.75.

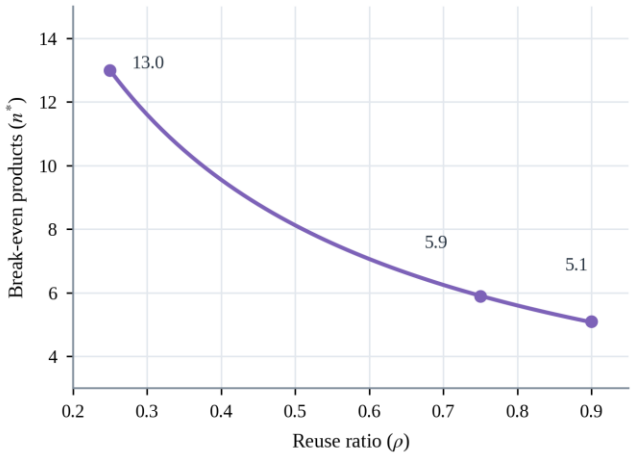


Fig. 6. Sensitivity of the Break-Even Point to the Reuse Ratio

TABLE VI. Change Lead Time under Assumed Conditions

Change type	**Code-embedded (weeks)**	**Metadata-driven (weeks)**	**Reduction (%)**
Rate table update	6	1.5	75.0
New plan option	8	2	75.0
Eligibility rule	5	1	80.0
Approval threshold	4	0.5	87.5
Form template	4	1.5	62.5
New market launch	20	8	60.0
Total	**47**	**14.5**	**69.1**

Lead times are assumed values for illustration.

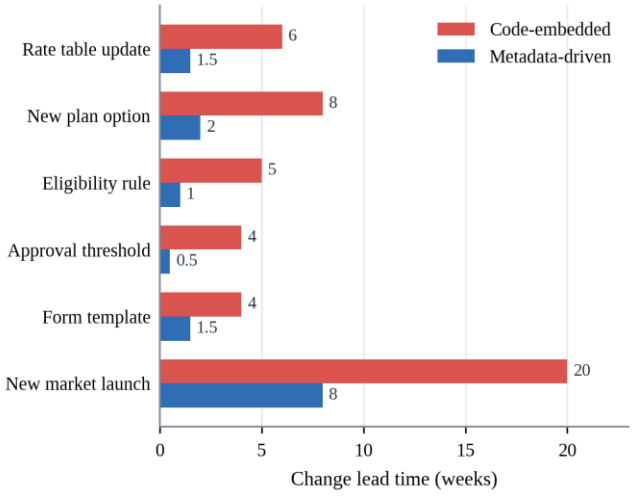


Fig. 7. Change Lead Time by Change Type under Both Approaches

7. Discussion

A number of factors, social, organizational and managerial, affect the success or failure of the industrial application of model-driven engineering, as experienced in practice [27]. A metadata layer thus must be owned, have review and approval processes for product changes, and be trained to the product specialists. If not, inconsistency in metadata can grow, making up for the anticipated effort savings. Table VII summarizes the principal risks and mitigations, such as metadata sprawl, runtime overhead, and loss of traceability.

TABLE VII. Principal Risks and Mitigations

Risk	Mitigation
Metadata sprawl and inconsistency	Constraint validation at authoring time; naming standards
Runtime interpretation overhead	Caching of compiled metadata; selective pre-generation
Regulatory traceability	Effective-dated versions; dedicated audit block
Weak governance of ownership	Role-based authoring; review workflow
Interface drift between blocks	Contract tests; versioned service contracts

The evolution of microservices has brought along a number of complex issues such as distributed data management, system complexity, and distributed service testing [28]. The same concerns are anticipated for standalone deployment of blocks, and there is a push towards testing the block interfaces, versioned metadata releases, and observability of quote transactions. Regulatory requirements are also supported by audit trails and effective dated metadata which enables the actual rules that were applied to a past quote to be reproduced.

Design science contributions may occur at various stages of the problem and the solution's development, from routine design to improvement, invention, and exaptation [29]. The proposed architecture can be seen as an exaptation, because the concepts of product line engineering, model driven engineering, and service orientation – which are already available in the domain of software – are transferred to the insurance sales domain. Some limitations are that it is an analytical evaluation and the parameters are assumed. Extensions to the catalogue to claims and servicing journeys, pilot deployments, and automated analysis of insurance feature models are directions for further work.

9. Conclusion

By using a metadata layer to define and configure the building blocks, reusable digital building blocks provide a structured solution to the duplication and maintenance needs of health insurance sales applications. The synthesis of the 29 sources confirmed that a layered architecture in which there is separation between business rules and application code is supported by modularity, product line variability, model-driven engineering, service orientation, and process management.

The results of the analytical evaluation were a break-even point at 6 products, an effort reduction of 37.2% at 12 products, and a change lead time reduction of 69.1%. The quantitative results have yet to be empirically validated in operational insurance environments, before which they can only be generalized.

REFERENCES

1. M. Eling and M. Lehmann, “The impact of digitalization on the insurance value chain and the insurability of risks,” *Geneva Papers Risk Insurance—Issues Pract.*, vol. 43, no. 3, pp. 359–396, 2018, doi: 10.1057/s41288-017-0073-0.
2. E. Stoeckli, C. Dremel, and F. Uebernickel, “Exploring characteristics and transformational capabilities of InsurTech innovations to understand insurance value creation in a digital world,” *Electron. Markets*, vol. 28, no. 3, pp. 287–305, 2018, doi: 10.1007/s12525-018-0304-7.
3. D. Tilson, K. Lyytinen, and C. Sørensen, “Digital infrastructures: The missing IS research agenda,” *Inf. Syst. Res.*, vol. 21, no. 4, pp. 748–759, 2010, doi: 10.1287/isre.1100.0318.

4. S. Nambisan, K. Lyytinen, A. Majchrzak, and M. Song, “Digital innovation management: Reinventing innovation management research in a digital world,” *MIS Quart.*, vol. 41, no. 1, pp. 223–238, 2017, doi: 10.25300/MISQ/2017/41:1.03.
5. A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information systems research,” *MIS Quart.*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.
6. K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *J. Manage. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.
7. D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972, doi: 10.1145/361598.361623.
8. R. Sanchez and J. T. Mahoney, “Modularity, flexibility, and knowledge management in product and organization design,” *Strategic Manage. J.*, vol. 17, no. S2, pp. 63–76, 1996, doi: 10.1002/smj.4250171107.
9. C. W. Krueger, “Software reuse,” *ACM Comput. Surveys*, vol. 24, no. 2, pp. 131–183, 1992, doi: 10.1145/130844.130856.
10. W. B. Frakes and K. Kang, “Software reuse research: Status and future,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 529–536, 2005, doi: 10.1109/TSE.2005.85.
11. K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005, doi: 10.1007/3-540-28901-1.
12. D. Batory, “Feature models, grammars, and propositional formulas,” in *Software Product Lines*, H. Obbink and K. Pohl, Eds. Springer, 2005, pp. 7–20, doi: 10.1007/11554844_3.
13. D. Benavides, S. Segura, and A. Ruiz-Cortés, “Automated analysis of feature models 20 years later: A literature review,” *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, 2010, doi: 10.1016/j.is.2010.01.001.
14. Y. Yoo, O. Henfridsson, and K. Lyytinen, “Research commentary—The new organizing logic of digital innovation: An agenda for information systems research,” *Inf. Syst. Res.*, vol. 21, no. 4, pp. 724–735, 2010, doi: 10.1287/isre.1100.0322.
15. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-oriented computing: State of the art and research challenges,” *Computer*, vol. 40, no. 11, pp. 38–45, 2007, doi: 10.1109/MC.2007.400.
16. N. Dragoni *et al.*, “Microservices: Yesterday, today, and tomorrow,” in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Springer, 2017, pp. 195–216, doi: 10.1007/978-3-319-67425-4_12.
17. D. C. Schmidt, “Model-driven engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, 2006, doi: 10.1109/MC.2006.58.
18. S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley, 2008, doi: 10.1002/9780470249260.
19. M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Comput. Surveys*, vol. 37, no. 4, pp. 316–344, 2005, doi: 10.1145/1118890.1118892.
20. K. Czarnecki and S. Helsen, “Feature-based survey of model transformation approaches,” *IBM Syst. J.*, vol. 45, no. 3, pp. 621–645, 2006, doi: 10.1147/sj.453.0621.
21. A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices architecture enables DevOps: Migration to a cloud-native architecture,” *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, 2016, doi: 10.1109/MS.2016.64.
22. M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2013, doi: 10.1007/978-3-642-33143-5.
23. M. Weske, *Business Process Management: Concepts, Languages, Architectures*, 2nd ed. Springer, 2012, doi: 10.1007/978-3-642-28616-2.
24. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, “Workflow patterns,” *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003, doi: 10.1023/A:1022883727209.
25. P. Mohagheghi and R. Conradi, “Quality, productivity and economic benefits of software reuse: A review of industrial studies,” *Empirical Softw. Eng.*, vol. 12, no. 5, pp. 471–516, 2007, doi: 10.1007/s10664-007-9040-x.
26. M. M. Lehman, “Programs, life cycles, and laws of software evolution,” *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.
27. J. Hutchinson, J. Whittle, and M. Rouncefield, “Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure,” *Sci. Comput. Program.*, vol. 89, pp. 144–161, 2014, doi: 10.1016/j.scico.2013.03.017.
28. P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, 2018, doi: 10.1109/MS.2018.2141039.
29. S. Gregor and A. R. Hevner, “Positioning and presenting design science research for maximum impact,” *MIS Quart.*, vol. 37, no. 2, pp. 337–355, 2013, doi: 10.25300/MISQ/2013/37.2.01.