# Technical Report

## Towards Test-Driven Model Development in Production Systems Engineering

Laura Waltersdorfer[1,2]
Felix Rinker[1,2]
Stefan Biffl[2]

[1] Christian Doppler Laboratory for Security and Quality Improvement in the Production System Life Cycle, Institute of Information Systems Engineering, TU Wien, Vienna, Austria
<firstname>.<lastname>@tuwien.ac.at

[2]TU Wien, Institute of Information Systems Engineering, Vienna, Austria
<firstname>.<lastname>@tuwien.ac.at

Technical Report No. CDL-SQI 2020-01

Issued: January 2020

Citation: L. Waltersdorfer, F. Rinker, S. Biffl "Towards Test-Driven Model Development in Production Systems Engineering", Submitted to 22nd International Conference on Enterprise Information Systems (ICEIS), Technical Report CDL-SQI 2020-01, TU Wien, Vienna, Austria, January 2020.

# Towards Test-Driven Model Development in Production Systems Engineering

Laura Waltersdorfer[1,2][a], Felix Rinker[1,2][b] and Stefan Biffl[2][c]
[1]*Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), Technische Universität Wien, Austria*
[2]*Institute of Information Systems Engineering, Technische Universität Wien, Austria*
*{firstname}.{lastname}@tuwien.ac.at*

Keywords: Test-Driven Model Engineering, Production Systems Engineering, Meta-Model Engineering, Testing Pipeline, Test-Driven Model Development, experience report

Abstract: The explicit representation of discipline-specific knowledge in manufacturing contexts is becoming more important due to rising competition, higher system complexity and personnel changes. However, domain experts do seldom have sufficient technical and theoretical knowledge or adequate tool support required for productive and effective model engineering. Test-driven modeling, similar to test-driven software engineering can support an effective and efficient model engineering process to produce high-quality meta- and instance models by incorporating consistency and semantic checks during the model engineering. We report on our experiences from an industry project with a company partner and present a testing pipeline for production system use cases supporting the modeling of discipline-specific AutomationML instance models. Debug output for the models is generated to support non-technical engineers in the error detection of discipline-specific models. For future work user-friendly test definition is in planning.

## 1 INTRODUCTION

Model Development and Engineering have become an increasingly important activity for the development of cyber-physical production systems (Berardinelli et al., 2016): Software is used to increase the throughput and efficiency of such systems, while ensuring security and safety. One major use cases for software engineering in the manufacturing domain are the simulation engineering, which aims to simulate the real world behavior of all integrated components to reduce defects, errors in the design and miscalculations (Košturiak and Gregor, 1999). In order to get this integrated view, inputs from all relevant, involved engineering disciplines is needed.

The second use case is the automation engineer, writing control code for components, such as conveyors or robot arms to enable efficient production. Both roles need an integrated view on the planned components and the overall view on the production system to write and model appropriate code and models.

Regarding the data exchange, software also impacted the manufacturing domain heavily: Paper plans used for planing and modeling of production systems were over time replaced by exchanging digital artefacts, describing the different views needed for planning production systems (Madsen and Munck, 2017): For example, the mechanical view describes the layout of plants, the electrical plan describes the wiring and power supply and the fluidics, describing the different liquids required for the system. In 2009, Ebert describes that the complexity of embedded systems grows nearly exponentially (Ebert and Jones, 2009), since the combination of hardware and software adds another layer to the overall complexity of such systems.

However, proprietary formats, heterogenous views and non-suitable tools common to production system engineering can lead to inconsistent naming and tagging of concepts or also hinder semantic or project-independent consistency (Feldmann et al., 2019). Best practices or industry guidelines (for example (Verlag, 2009) (Schüller et al., 2019)) are not

[a] https://orcid.org/0000-0002-6932-5036
[b] https://orcid.org/0000-0002-6409-8639
[c] https://orcid.org/0000-0002-3413-7780

adhered due to the unstructured process of describing domain-specific knowledge (Biffl et al., 2019). However, such standards would be essential for automated testing regarding security and correctness of engineering models and plans.

To alleviate some of these shortcomings, the authors in (Munck and Madsen, 2015), propose Test-Driven Modeling (TDM) analogue to Test-Driven Development (TDD), to ensure increased quality of engineering models. TDD is a well-established method from Software Engineering (SE) to improve the overall quality of the development process by incorporating continuous testing of specific test use cases throughout the writing of software code.

This paper is an experience report on an improvement initiative of a industry research project, to prove the suitability and functioning of test-driven model engineering for domain-specific models in AutomationML (AML) and describe our lessons learned from a prototypical application.

Instead of manual integration of local views and the tedious finding of errors, the automation of test reports has the potential to increase consistency and flexibility of instance models and prevent common pitfalls, such as non-unique ids or missing, non-retrievable links. Furthermore, our approach can add flexibility to the modeling process by introducing an iterative process, enabling stakeholders to add additional disciplines and thus, additional complexity to the models.

The remainder of this paper is structured as follows: Section 2 presents related work on model-based approaches, domain-specific model engineering in production systems engineering and test-driven (model) development. Section 3 presents identified challenges in model development and engineering in production systems engineering. In section 4, we introduce the solution approach and present preliminary results of the test-driven model development. We discuss our findings and related limitations in Section 5. Section 6 summarizes and identifies future work.

# 2 RELATED WORK

This section gives an overview about related work on model-based approaches, domain-specific model engineering and TDD.

## 2.1 Model-Based Approaches

Model-based approaches in software engineering and other contexts is said to improve modularity, reusability of components and technology flexibility (Brambilla et al., 2017). However, in engineering spreadsheets are common tools and popular formats are Portable Document Format (PDF) and Comma Separated Value (CSV) for designing (data) models and for encoding of essential information for other stakeholders due to their minimal start-up time. These tools are not suitable anymore for the description of engineering knowledge due to growing complexity of production systems and such environments can pose significant challenges (Madsen and Munck, 2017). Unified Modeling Language (UML), as a popular language in software engineering has been proven to be applicable for automated production systems as well (Vogel-Heuser, 2014). Another challenge in the production systems engineering context is the heterogeneity of tools and views, which AML aims to bridge by providing a data format designed for engineering data exchange (Drath et al., 2008). However, there are still multiple challenges associated to model-based approaches in the industry: Integrated views on the local views, linking support for common concepts and inconsistency management are major issues (Feldmann et al., 2019).

## 2.2 Domain-Specific Model Engineering in Production Systems Engineering

With the digitisation of the manufacturing domain and increasing world wide competition, the need for modeling engineering in production systems engineering is increasing (Berardinelli et al., 2016). However, a well-known problem is that although domain experts have extensive domain-specific knowledge inside their minds based on years of experiences and through multiple cases seen in their career, they often cannot explicitly model this essential information in correct and effective ways. Unfortunately, domain experts are seldom modeling experts, and are not aware about the importance of adhering to formal constraints and formalisation of modeling concepts (López-Fernández et al., 2015), which are necessary for automatic testing, consistency checks and more. Traceabilty of design decisions is a major factor for sustainable engineering knowledge documentation (Kathrein et al., 2019). Thus, domain experts require the assistance of tools or also engineers to construct suitable models.

Domain-specific models used for software engineering are described in domain-specific modeling languages, which is suitable for the concepts and application. Meta-models are usually required to design instance models of real world scenarios and to check for consistency according to the constraints (López-Fernández et al., 2015).

In figure 1, the manual (meta)-modeling process common to production systems engineering is illustrated: Different disciplines, plant, mechanical, fluidic and electric planning are working in parallel on various artefacts, also called *local models*, describing his discipline-specific view and concepts. The other stakeholders then need to extract the information into their local models and incorporate the changes by others to see if the changes affected their own local concepts (Lüder et al., 2019). Stakeholders, who need an integrated view on the system, such as project managers, simulation and automation engineers, need integrated models, incorporating all relevant disciplines and their dependencies (Lüder et al., 2019).

For the further validation and consistency checking, artefacts need to be sent around and are manually analysed. Since such artefacts and models can be big and are hard to read for humans due to nested structure and no automated tests, errors are highly probable.

### 2.3 Test-Driven (Model) Development

The main paradigm of TDD is to write tests before composing software code (Astels, 2003). This way, TDD can increase the quality of output of the software engineering process dramatically: Advantages are increased efficacy and feedback, catching defects and bugs in early phases and shipping maintainable and tested code (Williams et al., 2003) (Erdogmus et al., 2005). Microsoft showed increased productivity of development teams by applying TDD methodology even taking in account extra upfront effort (Bhat and Nagappan, 2006).

The methodology can also be applied to modeling as described by Zhang (Zhang, 2004). Munck et Madsen propose a test-driven model-based systems engineering method to test architecture and behavior of models in (Munck and Madsen, 2015). In (Munck and Madsen, 2017) they also report on their experiences utilising formal verification, simulations and forecasting to support and enable the development of cloud-based complex medical systems. (Zolotas et al., 2017) shows that the error rate with constraint testing can be dramatically reduced for certain errors in comparison to manual checking.

## 3 RESEARCH METHODOLOGY

Vogel-Heuser et al. have applied the concept of technical debt to the production systems domain (Vogel-Heuser et al., 2015): Technical debt is a concept from software engineering, to quantify and model design and implementation shortcomings from industry or other standards in technical environments due to time constraints or other reasons. Biffl et al. have analysed technical debt in the data exchange of production systems engineering (Biffl et al., 2019) within the context of a case study. The applicability of technical debt in the industrial domain motivates the transition of software engineering methods (in our case TDD) to the production system engineering domain to manage the negative effects and increased costs from accumulated technical debt.

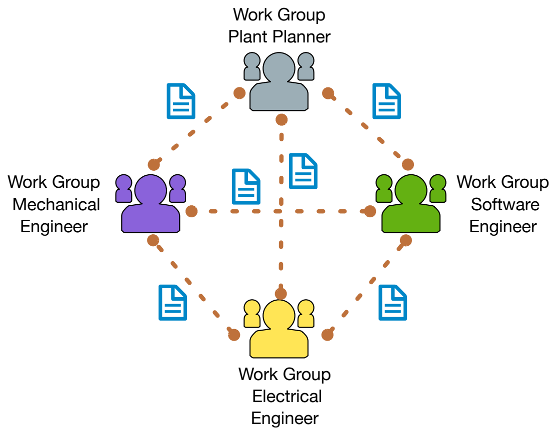

Figure 1: Manual meta-modeling process

In the context of technical debt management and transfer of software engineering methods to the production system domain (Vogel-Heuser et al., 2015), we designed a conceptual and prototypical approach to implement test-driven model development for the production systems context. We report on our experiences and lessons learned from the context of an applied research project over eight months: The context is an improvement initiative with a company partner, in the manufacturing domain to improve the engineering data management process and data model lifecycle. Through multiple workshops, we elicited stakeholder needs on domain-specific models and the workings of data exchange in production systems engineering [1].

## 4 SOLUTION APPROACH

In this section we discuss our solution approach, a test pipeline for instance models of domain-specific instance models of engineering disciplines common to the production systems engineering domain.

[1] sources can be provided

## 4.1 Context and Architecture

The basic use case for our conceptual test-driven model development approach is a so-called *project configuration* for a production systems engineering project, providing the initial structure of a project. A work cell or entire plant is planned or re-configured according to customer input.

We have three roles representing traditional engineering disciplines in such project, also introduced in figure 1: Plant planner providing the general structure of the cell or plant, mechanical engineer responsible for the mechanic components, fluidic engineer responsible for all kind of liquids and electric engineer designing wiring and power supply.

Similar to the process of constructing metamodels in (López-Fernández et al., 2015), one of our authors acted as the engineer and constructed a metamodel for the production systems domain in collaboration with a domain expert. The domain expert is an consultant for our research project, with extensive experience in industrial projects and research in academia. Over the course of eight months, both stakeholders were iteratively designing the architecture and testing pipeline, adapting it to real-world requirements provided by partner company stakeholders during the industry research project. Within the context of this project, tests were added when errors became evident and hindered the progress.

In figure 2, our testing pipeline is illustrated: The overall architecture is based on service-orientated architecture to build data exchange models to enable engineering data exchange in production systems engineering between different disciplines.

A meta-metamodel is described in a YAML Ain't Markup Language (YAML) file, a data serialisation language, which provides the general structure for the project context. According to the Meta Object Facility (MOF) hierarchy, this represents layer M3. Based on this configuration context, user models, AML-1 and AML-2 templates are generated automatically. AML-2 instance models would be the representation of the real world according to single disciplines, for example the electrical floor plan. AML-1 models are the integrated view, which provide a holistic overview over all relevant disciplines. The templates provide the structure how the models can be modelled, which disciplines can be integrated and they are verified in the next step. Verification tests that take place here, are of semantic and syntactic nature. The successful passing of the verification tests means that the templates are well-formed within the context of AML. In case of semantic or syntactic errors, an report is generated to improve bug tracking and fixing of errors. In case, no errors were detected, the transformer then populates the AML-2 template with data based on both AML-1 and AML-2 templates, and on data provided by the engineering discipline-specific tools. After the template has been filled, another set of tests is conducted to validate parameters, relationships and dependencies based on previously given constraints and tests put into a constraints testfile. An example for such a constraint can be that the speed of a conveyor must not exceed a certain limit depending on the product. The result is either a valid AML-2 model, that is well-formed according to the meta-metamodel or an error report with bug output for further adaptations.

## 4.2 Debug Output and Used Technologies

In listing 1, an example for debug output is depicted. The output shows several messages, explaining the processing of propagation of links throughout the different components of the models. Errors are shown in red, in our case the first element that was linked to a sub concept was not retrievable by the system.

Advantages of debug output and the overall methodology are that project-dependent changes can be made traceable and retrievable if change logs or error reports are versioned. Furthermore, the integrated view is validated and tested based on self-designed tests to verify the correctness and adherence to certain constraints. Thus, this approach can support engineers to produce faster, reliable models for production systems engineering. The manual error detection, which tends to be cumbersome for non-technical domain experts, can be supported by automated test reports, which can be then resolved line by line.

YAML , a successor of Extensible Markup Language (XML), is a human-readable description language, making it easier for non-technical domain experts to read the models. AML as a new standard in the production system domain has extensive potential to simplify and support the heterogeneous data landscape common for this domain. The usage of this standard might enable the future implementation of other AML and tool interfaces. The decision to not use Ecore, and Eclipse Modeling Framework (EMF), popular for model engineering is the independence from the Eclipse workspace. Furthermore, EMF offers more features than needed for our use case, and we wanted to keep the complexity low within the project.

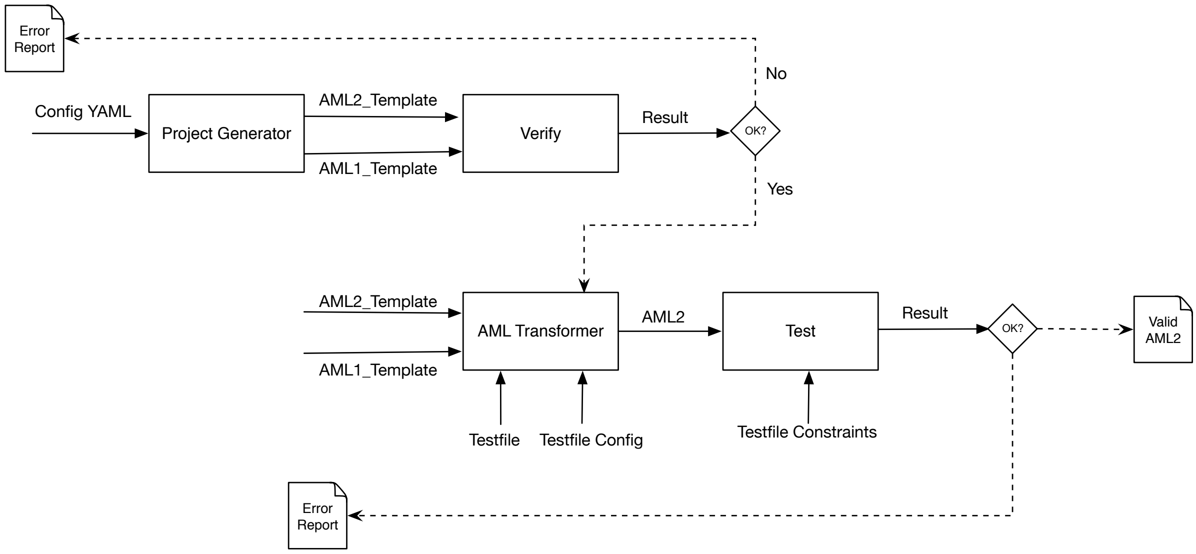


Figure 2: Architecture of proposed testing pipeline

Listing 1: Debug output

```
DEBUG - Found propagation mapping 'aml://AML1Project#this.FunctionView@FunctionViewID' for 'SimulationView@MUK'
DEBUG - Extracted Mapping Path: this.FunctionView@FunctionViewID
DEBUG - Found SubConcept 'FunctionView' in Concept 'K20_50_D_15_BE10_MKL'
ERROR - Could not find Attribute 'FunctionViewID' in SubConcept 'FunctionView'
ERROR - Error resolving attribute 'FunctionViewID' for path 'this.FunctionView' in element 'K20_50_D_15_BE10_MKL'
ERROR - Error processing attribute path 'aml://AML1Project#this.FunctionView@FunctionViewID'
DEBUG - Found propagation mapping 'aml://AML1Project#this.FunctionView@Description' for 'SimulationView@REMARK'
DEBUG - Extracted Mapping Path: this.FunctionView@Description
DEBUG - Found SubConcept 'FunctionView' in Concept 'K20_50_D_15_BE10_MKL'
DEBUG - Found Attribute 'Description' in Element 'FunctionView'
DEBUG - Found propagation mapping
```

# 5 DISCUSSION

The use case starting with only three disciplines is realistic to the set-up phase of real, industrial production system engineering projects. However, one limitation of our approach is that the simple use case was only simulated with the example set up and no additional disciplines were added to the context during the project until now.

Nevertheless, the advantage of our approach is that in future, additional disciplines could be added relatively easy throughout the process: In case of changes to the meta-model and additional instance models, the previously passed tests have to be re-run according to the TDD paradigm. This way, checks are conducted again and ensure the consistency and validity of the local and the integrated models in the new context although changes have been committed. Another limitation is the limited set of test so far, only detected errors were so far implemented into the testing pipeline.

Noteworthy, is also that up time effort is certainly increased for the setup of the testing pipeline: Domain experts and engineers have to collaborate and externalise implicit knowledge into meta-metamodel, the domain-specific language templates and the constraints testfiles. However, during our improvement initiative we already observed positive benefits as easier bug tracking and error reporting, as well as the reduction of common mistakes such as non-unique identifiers or missing subconcepts. We assume that similar to (Bhat and Nagappan, 2006), the additional set up effort is similar in such a specialised domain as production systems engineering as in the industrial case studies. However, the benefits of such a TDD approach can increase the productivity of domain experts and technical stakeholders immensely.

# 6 CONCLUSION AND FUTURE WORK

Consistency and traceability are major issues in the production systems engineering domain and in the representation of engineering knowledge and expertise. Due to various disciplines and the diverging tool and format landscape, consistency and other checks are tedious and error-prone, if conducted manually. Domain experts often are not modeling experts, and therefore are not able to test their domain models systematically. Thus, we have presented a testing pipeline to support discipline-specific model engineering in the production systems engineering domain. Through an iterative process with a domain experts, an industrial use case and an experienced model engineer, we designed an appropriate architecture and models, showing real-world application of test-driven model engineering methodology. The error reports simplified the communication with the domain expert to convey issues in the models, and also the resolving of such issues. Although, we used specific technologies, our service architecture can be used as a base model for other applications and prototypes in this field. The results are promising to extend the application of our approach and to measure its impact with industrial partners in the future. For future work, the extension to additional disciplines and other models need to be done. Furthermore, the usability of our solution should be also extended, since the configuration and implementation is currently done via bash scripts and the implementation of new tests only available to experienced engineers.

## ACKNOWLEDGEMENTS

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital & Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

## REFERENCES

Astels, D. (2003). *Test driven development: A practical guide*. Prentice Hall Professional Technical Reference.

Berardinelli, L., Biffl, S., Lüder, A., Mätzler, E., Mayerhofer, T., Wimmer, M., and Wolny, S. (2016). Cross-disciplinary engineering with automationml and sysml. *at-Automatisierungstechnik*, 64(4):253–269.

Bhat, T. and Nagappan, N. (2006). Evaluating the efficacy of test-driven development: industrial case studies. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 356–363. ACM.

Biffl, S., Ekaputra, F., Lüder, A., Pauly, J., Rinker, F., Waltersdorfer, L., and Winkler, D. (2019). Technical debt analysis in parallel multi-disciplinary systems engineering. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 342–346. IEEE.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 3(1):1–207.

Drath, R., Luder, A., Peschke, J., and Hundt, L. (2008). Automationml-the glue for seamless automation engineering. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 616–623. IEEE.

Ebert, C. and Jones, C. (2009). Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52.

Erdogmus, H., Morisio, M., and Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. *IEEE Transactions on software Engineering*, 31(3):226–237.

Feldmann, S., Kernschmidt, K., Wimmer, M., and Vogel-Heuser, B. (2019). Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering. *Journal of Systems and Software*, 153:105–134.

Kathrein, L., Lüder, A., Meixner, K., Winkler, D., and Biffl, S. (2019). Product/ion-aware analysis of multi-disciplinary systems engineering processes. In *21th International Conference on Enterprise Information Systems (ICEIS 2019)*.

Košturiak, J. and Gregor, M. (1999). Simulation in production system life cycle. *Computers in Industry*, 38(2):159–172.

López-Fernández, J. J., Cuadrado, J. S., Guerra, E., and De Lara, J. (2015). Example-driven meta-model development. *Software & Systems Modeling*, 14(4):1323–1347.

Lüder, A., Pauly, J.-L., Rinker, F., and Biffl, S. (2019). Data exchange logistics in engineering networks exploiting automated data integration. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 657–664. IEEE.

Madsen, J. and Munck, A. (2017). A systematic and practical method for selecting systems engineering tools. In

*2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE.

Munck, A. and Madsen, J. (2015). Test-driven modeling of embedded systems. In *2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC)*, pages 1–4. IEEE.

Munck, A. and Madsen, J. (2017). Test-driven modeling and development of cloud-enabled cyber-physical smart systems. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE.

Schüller, A., Modersohn, A., Kawohl, M., and Wrede, J. (2019). Der digitale zwilling in der prozessindustrie. *atp magazin*, 61(1-2):70–81.

Verlag, V. (2009). Verein deutscher ingenieure: Vdi richtlinie 3695–engineering von anlagen–evaluieren und optimieren des engineerings. *German industrial recommendation*.

Vogel-Heuser, B. (2014). Usability experiments to evaluate uml/sysml-based model driven software engineering notations for logic control in manufacturing automation. *Journal of Software Engineering and Applications*, 7(11):943.

Vogel-Heuser, B., Rösch, S., Martini, A., and Tichy, M. (2015). Technical debt in automated production systems. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 49–52. IEEE.

Williams, L., Maximilien, E. M., and Vouk, M. (2003). Test-driven development as a defect-reduction practice. In *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pages 34–45. IEEE.

Zhang, Y. (2004). Test-driven modeling for model-driven development. *Ieee Software*, 21(5):80–86.

Zolotas, A., Clarisó, R., Matragkas, N., Kolovos, D. S., and Paige, R. F. (2017). Constraint programming for type inference in flexible model-driven engineering. *Computer Languages, Systems & Structures*, 49:216–230.