Proceedings of 15th International Marine Design Conference (IMDC-2024)
June 2-6, 2024
Amsterdam, the Netherlands

Practical implementation of configuration management in the context of concept ship design – first lessons

S. Bedert[1,*], R. Hoogenboom[2]

ABSTRACT

The implementation of configuration management (CM) in support of concept design activities at the Products and Proposals department of Damen Naval has put forward a number of challenges that seem to be unique to the shipbuilding industry and the process of concept development therein, that traditional enterprise-level product lifecycle management software and processes are unable to tackle. Information fluidity, an uneven and fast evolving tooling landscape and limited team sizes have driven us to pursue dedicated software implementations in support of CM during concept design. From the first attempts of doing so, we have seen that a number of performance requirements for the software that were completely under the radar are now driving factors for further selection (mostly related to speed and agility of the tool), while some others even impact our way of working and organizational structure as a whole, requiring us to carry system responsibility per discipline, instead of only functional responsibility.

KEY WORDS

Configuration Management; concept design; contract design; database; shipbuilding; ship design;

INTRODUCTION

This paper aims to describe the lessons learned derived from our initial attempts to introduce configuration management and the associated tooling in our concept design workflow at the Proposals department of Damen Naval. Given that the number of theoretical papers on the subject, and descriptions on cross-industry abstraction levels are widespread, the explicit premise of this report is to stay as practical and 'down-to-earth' as possible, noting and highlighting primarily the challenges faced within the context of our own department, and the naval shipbuilding industry by extension, as far as the processes and organizational structures applied are comparable with those of Damen Naval (which cannot be verified by the author).

The paper starts by defining configuration management (CM) as applied in the context in this paper and the reasons why this is deemed 'special' in the context of ship concept design. In a second part, the paper shortly touches the rules of play of configuration management (CM) as laid out at Damen Naval and describes two initiatives rolled out for the introduction of configuration management software.
In the final part, the actual lessons learned from these initiatives are described, with a possible outlook for further work.

SCOPE DEFINITION

This chapter defines (and restricts) the scope of the term Configuration Management to that part relevant in the context of this paper and the concept design department of Damen Naval and describes why this is deemed to be a special case of implementation compared to more widespread descriptions, definitions and uses of the concept warranting custom implementation.

[1] Manager Marine Engineering at Damen Naval B.V. (Dept. Proposals; Vlissingen, Netherlands);
[2] Manager Product Development & Innovation at Damen Naval B.V. (Dept. RD&I, Vlissingen, Netherlands);
* Corresponding Author: s.bedert@damennaval.com

Submitted: 15 January, Revised: 29 April 2024, Accepted: 1 May 2024, Published: 23 May 2024

Conference paper, DOI: https://doi.org/10.59490/imdc.2024.887 e-ISSN: 3050-4864

Configuration Management

Since Configuration Management is a multifaceted concept, it is essential to establish precise boundaries for its application in the context of this paper.

The fundamental goal of the work done is/was to create a centralized repository for information related to the physical components and systems that constitute a ship, especially suitable for use during concept- and tender design phases in a ships lifecycle. This information, especially the aspects which holds cross-disciplinary importance, such as size, weight, and electrical power requirements, will be managed by the relevant specialists. The objective is to maintain this information as a constantly available and up-to-date snapshot over time. In simpler terms, we are focusing on managing the ship's system configuration, rather than the concept design project configuration, which primarily involves document and requirement management and has not been considered in this context.

The overall aim is to allow the different systems in the ship to be developed in accordance with their own largely unaligned, albeit overlapping, timelines, without having to wait for a full design cycle on all systems before an updated snapshot is made available, as long as all the data available is the ‘latest status available at that time’ in any given snapshot.

Once the initial idea was proposed, we structured it into an internally peer-reviewed 'master' functional specification (the concept was dubbed Krypton at the time, a name which persists to this day within our company for anything related to the concept). With this functional specification in hand, we pursued the following implementation routes:

a. We invited software vendors to offer solutions that align with the specification.
b. We developed software in-house from scratch that matches the specification.
c. We sought partnerships to explore and enhance an existing system based on this specification.

Over time, steps (a) through (c) have been carried out in overlapping cascade, with (a) now completed, (b) nearly finished/scrapped, and (c) in a start-up phase. As a result, this paper focuses on the lessons learned during steps (a) and (b), which are of course also being considered in the execution of step (c).

Concept Development/Contract Design vs Engineering

It is to be noted that configuration management both in terms of document control and technical data management are well established within the (ship) engineering and building processes. In support of these, many well-established (Product Lifecycle Management; PLM) software suites are available, both on an enterprise and small business level.

So, what makes ship concept design activities so much different from production activities that we claim we need special software to do it? The reasons for that as we have identified them within our company are the following:

- Data persistence rigidity: Concept design is largely a matter of design space exploration and solution generation. Configuration items are generated, changed and deleted again on the fly on every level of detail throughout the process. In opposition, data is created in a PLM environment once, and is made there to stay, because the underlying hard- or software is being procured, built or created in real life. In essence, a concept design is all about defining a solution and all (building) budgets related to it, while a production phase is about realizing a (pre-)defined solution from definition to reality as quickly and efficiently as possible.
- Interfacing to tools/availability of software functionality: The PLM packages in use are targeted for interfacing with the existing engineering and building tool suites (CAD/CAM packages etc.). Just as the PLM package itself, however, the intrinsic overhead of that tool suite is in almost all cases too high for use in concept design, and much simpler alternatives are used.
- Procedural overhead vs. project throughput times: the speed at which tenders, and concept development projects are started, scrapped, and renewed is not compatible with the application management overhead of traditional PLM packages. Traditionally, it can take up to 4 to 8 weeks to setup a project in our PLM environments, involving at least three different departments at an average of two people per department.

IMPLEMENTATION INITIATIVES

This chapter describes two implementation initiatives taken to introduce configuration management in our department, corresponding to cases (a) (implementation through external vendor software) and (b) (implementation through own software development) from the scope definition described above.

In each of these cases, the practical goal was to have:

- A collaborative piece of software that allowed multiple users to work simultaneously.
- Gather, per discipline, lists of equipment that contained sufficient data to support the three main ships balances (weight, electrical load and heat load) in a single database.
- Get all relevant data out of the database in a format that allowed updating of the balance sheets.

Although the goal seems quite simple, from a database point of view this proved to be quite the feat, as in practice, this implied the system had to be capable to manage a Product Breakdown Structure, a ship's space list with associated data and a ship's equipment list, and the allocation between those data concepts (e.g. a piece of equipment is always part of one or more parts in the PBS structure, and always allocated to a space).
Beyond that basic goal, ambitions where set even higher on a few functionalities. For example, we expected the data fields to be managed to be flexible (user-managed) and values to be flexible on unit of measurement. Furthermore, we wanted to be able to compare ship configurations (since it is applied during concept design), which meant we wanted to 'disable' part of the database temporarily in favor of an alternative (i.e. support for trade-off from within the tool analysis).

External Software (Case (a))

For the first pilot attempt, we selected a closed but flexible platform from a software vendor that offered a 'configured to spec' version of their software. The software had a principally closed architecture, with interfacing created especially for us to be able to implement data I/O using an excel interface. At the time of delivery, the software implemented approx. 80% of the original Krypton functional specification.

After initial testing of the software using parts of a fictional ship concept and a limited dataset, the application was deployed for use in the most recent large contract design project ran by the Damen Naval Proposal department to keep track of ship (system) configuration. The software was used over the full period of the project (p/m two years) and served as configuration management tool for disciplines Propulsion, Electrical, Auxiliaries and Automation.

Success criteria for the pilot execution did not constitute numerical KPI's as such. The sole, practical pass criterion was that the tool would enable multiple engineers to input data of equipment in their disciplinary scope and in accordance with their discipline's applicable timelines, and that data could be exported again in support of a weight calculation and electrical load balance, which it did to successful extent.

However, after careful review, it was still decided to discontinue its use, primarily due to the lessons learned *Adaptability* and *software performance*/*scaling challenges* as described below (even if these were not part of the pilot's original success criteria). The web portal providing the user interface, for example, became unmanageably slow due to the server capacity reserved for the proof-of-concept once the pilot project contained its 'full' list of equipment, spaces and PBS elements. As a further example, we were unable to change the data views ourselves for the most basic use cases (for example, showing the space number to which a piece of equipment was allocated in the tabular equipment overview was impossible without outside consultancy).

In-house Development (Case (b))

To implement the initial lessons learned concerning openness of interfaces to unlock data (refer to below), mostly instigated by server capacity limitations and a need for more customizability, we swung almost stereotypically to the opposite extreme of the spectrum and started building our own version of what Krypton should be, as opposed to an externally sourced, closed architecture software package. In that attempt, we built a rudimentary piece of software that implemented the core functionalities of the Krypton spec, using SQL (SQLite) databases and a Django framework with web front-end.

This software was never developed further than a proof-of-concept. It was never used in production, nor was it put through serious 'malicious' testing. Success criteria for this test were also not defined as numerical KPI's. The practical pass criterion here was to have a 'configurable' data model available (adaptable to different types of equipment and systems), in which we could define data and later extract it again, and in that goal, the software succeeded again without fail.

Although it could be considered a successful 'start', the work was never finished because we saw that the work involved would require us to venture too far off from our core competences, as described in the *Software Support* lesson learned described.

LESSONS LEARNED

The lessons learned from using the third-party software (case (a)) and programming our own tool(s) (case (b)), roughly fall into the following categories:

1. Software functionality and performance
2. Modelling conventions
3. Organizational structure
4. Obvious/novice mistakes

Firstly, we are not above admitting that some very basic mistakes were made in all this (category 4). As we do not buy, create, or configure software every day, we obviously fell into pitfalls which may be marked as stereotypical. To get them out of the way, the most notable:

- We grossly overestimated the percentage-ready associated with a proof-of-concept at approx. 80% instead of the 30% it deserved. As you will see from all the lessons learned below, we still have a lot to figure out. This applies independently to both implementation case (a) and (b) as some of the main lessons learned relate directly to business rules, and not software implementation itself.
- We underestimated the effect of computational speed and performance on useability and user satisfaction in our on-premise implementation, lapsing too far from well-established 0.1/1/10 norms posed by (Miller, 1968).

It is worthwhile to notice that the second point is directly related to the concept design challenges described in the Scope Definition chapter before. Given that data values are more fluid in concept design, the software speed associated to data manipulation should be an order of magnitude faster that what we have seen in the past in traditional PLM packages (where our company has more experience and a more widespread frame of reference). So software speed is a unique aspect in which we will expect more instead of less when compared to traditional PLM packages.

Software functionality and performance

Batch editing
In drafting the functional Krypton specification, the use of batch creation and editing was posed as a nice-to-have. However, in use of both Krypton implementations, this proved to be a more than essential feature. In hindsight, this is a perfectly logical conclusion provided that in reality, most of the data is created and/or updated in batches from parts of the design process (e.g. a space list is (re)generated from a CAD source in one go for all spaces in a ship; or an equipment list for a full system is defined by a specialist or generated by a subcontractor in one go). As such, the majority of interactions with users with the software will be batch related, and the functionalities provided thereto should be on-point.

Adaptability
Tooling in support of configuration management is not the only part of our working environment that is undergoing changes. With the launching of the Krypton concept, we also inspired a wave of initiatives that aim to disclose information between software tools, calculation sheets and 3D models regardless of what we come up with on the configuration management side (e.g. transporting the information from a 2D general arrangement to a 3D model for further evaluation).
With that, a whole new set of data and related data formats come into play that could also be relevant for plugging in to a CM database, signifying the importance of adaptability of the Krypton implementation, as the tools surrounding the CM platform will, with a high chance, evolve faster than the platform itself. This in turn poses a requirement for any Krypton platform to be highly adaptable, not only regarding input and output formats, but data visualization within the tool itself.

Scaling challenges
As the products we build are large in absolute scale by themselves, the number of elements within them that contribute significantly to their design balances is equally so. Consequently, the number of elements to be considered for CM is also large. Both of our implementation cases have shown that these numbers can cause problems in the tested software packages, which both rely on server-side processing for data integrity checks on each Create, Read, Update or Delete (CRUD) action on the data, when server-side processing capacity is not duly considered.

Apart from server-side computational power limitations, it has also been remarked that the lists and tabular data representations used on both our implementations started to become cumbersome to work with (with endless scrolling, and just-not-there-yet

filtering capabilities). Without knowing what to look for exactly for improvements in further experimentation, this is a point that deserves attention.

Modelling conventions

In a conceptual design phase, team sizes are notably smaller compared to a production environment, where an extensive group of engineers and CM professionals collaborate on ship configuration maintenance. In a 'typical' conceptual design team, each discipline is represented a single individual or a handful of people. These team members primarily engage in primary design activities, next to ensuring that the configuration management effort remains commensurate with the phase of development.

To prevent CM activities to dominate design work, it is essential to tailor the level of abstraction in the information provided to the specific requirements of the design phase. Consider a main engine as an illustrative example. The scope of delivery for a main engine extends beyond the engine itself. It encompasses a long list of associated elements, each with dedicated specifications regarding weight, dimensions, etc. (e.g., a preheater, a local control panel, or external coolers). However, during the conceptual design phase, the primary interest often only lies in confirming the presence of a main engine installation rather than scrutinizing the particulars of its constituent parts. Yet, if inexperienced engineers receive the instruction to model solely the main engine, there is a risk that they may overlook the associated components, inadvertently underestimating the overall system's weight. Nevertheless, it is impractical to model and maintain each of these individual components separately. Doing so would introduce superfluous data into the configuration, resulting in excessive effort and the potential loss of 'belongs-to' relationships among these objects.
Hence, it is imperative to permit aggregation of a configuration to the appropriate level, alleviating the burden of model maintenance while retaining sufficient details to instill confidence in users and model auditors regarding completeness and the preservation of 'belongs-to' relationships. For instance, one approach may involve modeling large items with three distinct weight values: one for the primary hardware, one for associated hardware fluids, and one for all auxiliary components.

However, as you can image, the aggregation level that is acceptable is not uniform over all the different systems in a ship, nor, if you are truly honest, across projects. Inevitably so, finding the correct aggregation level is a matter of getting a ‘feeling’, or more extensive experience in using such models. In the end, it is not unimaginable for this experience to end up in some sort of data modelling guideline.

Development time

Given the considerations mentioned, achieving proficiency in data aggregation, and identifying suitable data views requires substantial practice and experience. Given the rapid pace of concept design projects and their heavy workload, expecting rapid mastery is unrealistic.

It is important to leverage the flexibility inherent in any chosen tool and remain open to iterative adjustments. Nevertheless, crafting an effective working method will take considerable time, potentially spanning years. Therefore, exercising patience is crucial, particularly when encountering user complaints during the initial implementation phase.
However, it's essential to recognize that despite potential challenges, benefits from implementing configuration management in concept design can still be observed from the outset, even if perfection isn't immediately attainable.

Organisational Structure

Data Ownership

The introduction of configuration management has brought to surface a fundamental aspect in the organizational structure of various disciplines within our department, that requires attention and improvement to fully realize the potential of this initiative.

At our department, knowledge, responsibilities, and people are organized in traditional shipbuilding-related disciplines (naval architects, structures, propulsion, electrical…). That division in disciplines is, as the pilots done in the context of this paper have clearly shown however, purely on functional level, meaning that a specialist is held responsible for defining what is necessary of an item to perform its primary function, but not directly for any secondary information that is a direct consequence of that decision. For example, a mechanical engineer is responsible for defining the correct flow and pressure requirement for a main firefighting pump, but he is not held responsible or accountable for the weight and dimensions of these pumps.

That functional subdivision is a problem in terms of configuration management, as this requires, in practice, ownership of all item properties within a single owner, if only from an efficiency point-of-view. That is because, as you can imagine, the data source for information pertaining to a configuration item is concentrated at one place (for example, technical manual of a pump contains the QH-curve as well as a dimensional drawing). So, it would only be logical for a specialist to own all data related to that pump. Even more so, when considering that all that 'secondary' information will only change when the primary specialist makes a design change.

This requirement for workflow alignment entails that specialists within the department should start acting as general system owners instead of purely functional owners, whereby their responsibilities lie in all system parameters, not just the functional ones. This transition further demands a reevaluation of incentives to encourage specialists to take ownership across all system parameters. Finding the right motivators will be pivotal in driving this change, enabling a more efficient and cohesive configuration management approach.

Software Support Workload

Implementation case (b) has clearly highlighted the distinction between our expertise as shipbuilders and that of professional software developers. Our excursion into creating a proof-of-concept provided invaluable insights into the intricate dynamics of software development, and especially software maintenance.

Upon venturing into programming software intended for a broader user community, software that surpasses the complexities of typical online programming tutorials, we encountered the many challenges that software developers will surely recognize, such as outdated third-party source code packages, evolving library versions, deprecated functionalities, and the continual need for adaptation within our source code. Programming for a larger user base also highlighted the increased need for meticulous software documentation and ditto reliability. This underscored the necessity for extensive source code documentation to ensure sustained functionality, irrespective of the original creator.

In the end, it became apparent that continuing this trajectory would necessitate the establishment and ongoing maintenance of dedicated resources to support the tool we developed. This signified the integration of software development into the core activities of our department, which was, in hindsight never the intention.

CONCLUSIONS

Our first exercises with third-party and our own (attempt to create) software in support of configuration management has taught us that the road ahead of us to find that sweet spot where tools are at their peak of added value is going to be a long one still. Within a landscape where the tooling surrounding such attempts is also evolving rapidly, it is unavoidable that any attempt to build something sustainable will require a high level of flexibility and future adaptability to new interfaces.

Given that data sources are very dispersed during concept design, data entry will remain a matter of manual labor for the foreseeable future. As such, software performance in terms of speed and responsiveness is a key requirement for successful implementation of such.

Furthermore, the use of a configuration management concept has shown us that we can no longer rely on traditional functional ownership of systems across disciplines. Instead, we need to change our attitude towards data correctness to systemwide ownership, where specialists no longer care only for the functional performance of the systems they design, but also for every consequential aspect associated therewith.

Although the goals set before implementation cases (a) and (b) where largely met, the lessons learned described in this document have shown that continuing those paths is not preferable, yet the premise of configuration management for concept design does warrant further exploration.

Implementation case (c) as described above is the next iteration being considered. The views depicted in this paper will serve in a large part as additional boundary conditions, goals and requirements in that project. Implementation case (c) is seen as a hybrid between cases (a) and (b), whereby the aim will be to find a piece of software which is developed and managed by an external partner on the one hand (avoiding the maintenance burden of software identified in implementation case (b)), but which is, by design, sufficiently open to allow flexible data access and visualization (for example, one which provides a full-access external API that allows manipulating, importing and/or exporting or reading data using small scripts, or maybe a strong interface to MS Excel). At the time of writing this paper, implementation case (c) was in the initial start-up phase. Feedback from that case was thus not yet available for further evaluation.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN WRTITING

During the preparation of this work the author(s) used ChatGPT 3.5 to improve grammar and overall conciseness of writing style. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

CONTRIBUTION STATEMENT

S. Bedert: Conceptualization; writing – original draft. **R. Hoogenboom**: Supervision; writing – review & editing

ACKNOWLEDGEMENTS

First and foremost, the author would like to acknowledge Claassen, J.P. for his continued support in the exploration and implementation of this work within his department and the projects under his care. Further acknowledgement is made to the pilot user group for their patience and valuable user feedback for implementation scenario A; (de Jong, F.; Den Engelsman, J.; Ferea, V.; Lukasse, R.; Van de Voorde, L), as well as Schouten, G. for providing enthusiastic external feedback, proving that we are on (the right) track with the concepts we came up with. Further thanks goes out to N. Gheorghe for her hard work in programming our own efforts for CM as scenario B.

REFERENCES

Miller, R. (1968). Response time in man-compute conversational transactions. Proc. AFIPS Fall Joint Computer Conference Vol. 33, 267-277