INTERNATIONAL DESIGN CONFERENCE – DESIGN 2022
https://doi.org/10.1017/pds.2022.188

Through-Lifecycle Whole-Design Optimisation Using Software Deployment Toolchains

J. Gopsill [1,2,✉] and B. Hicks [1]

[1] University of Bristol, United Kingdom, [2] Centre for Modelling & Simulation, United Kingdom
✉ james.gopsill@bristol.ac.uk

Abstract

Through-Lifecycle Whole Design Design Optimisation is widely considered one of the future approaches to solving design problems featuring prominently in Model-Based Systems Engineering, Set-Based Design and Digital Twins. Yet, design optimisation remains siloed optimising for design subsets. In this paper, we review academic literature and a series of case studies to uncover the challenges in achieving Through-Lifecycle Whole Design Design Optimisation. This is followed by action research that has investigated the application of software deployment toolchains to overcome the challenges.

Keywords: design optimisation, optimisation, model-based systems engineering (MBSE)

1. Introduction

Through-Lifecycle Whole-Design Design Optimisation[1] is widely considered one of the next steps in solving design problems, featuring prominently in Model-Based Systems Engineering (MBSE), Set-Based Design and Digital Twins (Handawi, et al., 2021; LaSorda, et al., 2018). 'Whole-Design' refers to evaluating a design option against all relevant numerical models at a moment in the product's lifecycle, and 'Through-Lifecycle' refers to the ability to continually update the model ensemble to perform whole-design design optimisation.

The aim is deceptively simple, given a set of parameters that define the options in a design space, evaluate the options against the requirements to determine if they satisfy the constraints and score highly against the design objectives (Figure 1). This is achieved by engineers combining numerical models[2] developed throughout a product's lifecycle and presenting them to the computer to enumerate, calculate and solve for optimal design options. These are then reviewed by engineers, supporting them in making evidenced-based decisions on the design's direction. Advantages include greater exploration of the design space, objective (unbiased and unopinionated) evaluation of design options, sensitivity analysis, and extended parallel development of design options (set-based) through the design process.

However, application of design optimisation continues to be siloed with teams focusing on aspects, such as structural, topological, cost and manufacturability. This results in partial optimisation with conflicts in in determining the optimal design arising.

[1] E.g., Multi-Domain Optimisation, Multi-Objective Optimisation, Numerical Design of Experiments, Design Space Exploration, Multi-Objective Optimisation.

[2] E.g., Cost, Manufacturing and Assembly Processes, Computational Fluid Dynamics, Finite Element Methods, Multi-Physics, Electromagnetics, Thermofluid, Heat Transfer, Look-Up Tables, Machine Learning, Fatigue, Life Cycle Analysis and Kinematics. As well as different evaluate different scenarios, transient and static behaviour.

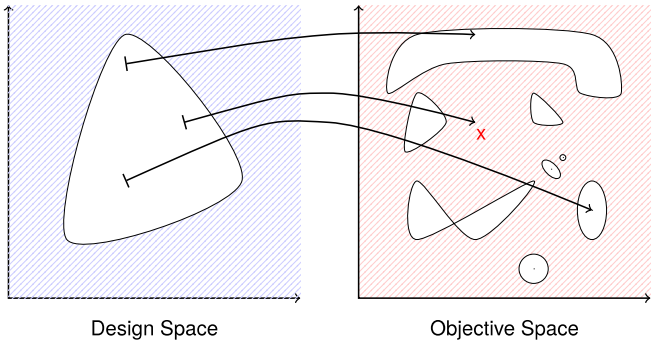


Figure 1. Through-Life Whole-Design Design Optimisation

Yet, industry's desire to perform Through-Lifecycle Whole-Design Design Optimisation continues to increase, driven by the ever-expanding transdisciplinary set of requirements introduced by initiatives such as, the Circular Economy, Design for X, sustainability, and decarbonisation. The initiatives have also led to a renewed focus to evaluate for different types of optimisation such as performance, robustness, and resilience. Further, organisations are increasingly concerning themselves with being able to describe the topology of a design space as well as visualising, querying, and navigating the design space. These capabilities support engineering designers in discussing design options and enabling evidence-based decision-making.

The lack of Through-Lifecycle Whole-Design Design Optimisation is also surprising given the extant academic research on the subject, degree of theoretical underpinning and the increasing number of industry-ready design optimisation toolsets[3] (Songqing & Wang, 2004; Stump, et al., 2007; Crowley, 2013). Thus, it is argued that the challenges must lie in the practical realisation and implementation of design optimisation with accounts reporting that it is both challenging and time-consuming leading to a subset of numerical models, which are of the same fidelity, domain[4], and featuring the same computational requirements[5], being used.

While the great degree of diversity poses challenges for existing approaches to configuring and orchestrating Through-Lifecycle Whole-Design Design Optimisation, there are a number of opensource practices that are being used in software development and deployment that could aid us in overcoming them. Practices, such as containerisation and continuous integration and continuous deployment (CI/CD), enable software developers to build modular architectures featuring a mixture of languages and software packages with reduced fear of conflicts and inter-operability issues.

Therefore, the contribution of this paper is in eliciting the practical challenges in achieving Through-Lifecycle Whole-Design Design Optimisation and how modern opensource software deployment practices could be used to overcome them. The paper achieves this by reviewing design optimisation review papers and recent applications of design optimisation performed by colleagues and industry partners to elicit the challenges (Section 2). This is then followed by a solution that utilises opensource software deployment approaches to overcome the challenges (Section 3). A practical demonstration is then presented (Section 4) with the results discussed alongside future areas of work (Section 5). The paper then concludes by highlighting the key findings from the study (Section 6).

2. Challenges in the Practical Implementation of Design Optimisation

To elicit the practical challenges in implementing design optimisation, the authors performed a semi-structured review of recent literature (>2017) via a Google Scholar and Engineering Design specific

[3] E.g., Altair HyperStudy, ANSYS DesignXplorer, Siemens HEEDS MDO, ESTECO modeFrontier, MSC Nastran Multi-Run & DSE, Noesis Solutions Optimus, Dassault Systémes Isight, Dynamo, MatLab, iChrome Nexus and Datadvance pSeven.

[4] E.g., a set of FEM simulations.

[5] E.g., OS, software, and CPU/GPU/storage.

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press

journals search concerning “design optimisation” and “design optimisation review”. This is complemented with recent case studies published by colleagues and industry partners who have performed design optimisation thereby representing a cross-section of practice to date.

2.1. Reviews on Design Optimisation

A review of design optimisation for preference modelling revealed six challenges in the practical application in the domain and were (Wang, et al., 2017):

1. adaptation of generic formulas to the specific domain problem;
2. incorporating expert opinion;
3. identifying the relationships between input and output parameters;
4. benchmarking and thresholding;
5. comparing design optimisation and expert solutions given the same time to assess RoI; and,
6. visualisation of the design space the design optimisation has mapped.

A further review of 98 academic and industry applications revealed many still combat <10 objective problems, which is still short of the 100/1000s of requirements designers need to manage (Mane & Narasinga Rao, 2017). Mane & Narasinga also highlight that there are few real-world applications compared to the number of hypothetical reference problems. A review of design optimisation in the water sector highlights four areas (Mala-Jetmarova, et al., 2018):

1. model inputs (formulating the representation of the design space);
2. algorithm & solution methodology;
3. search space and computational efficiency; and,
4. solution post processing.

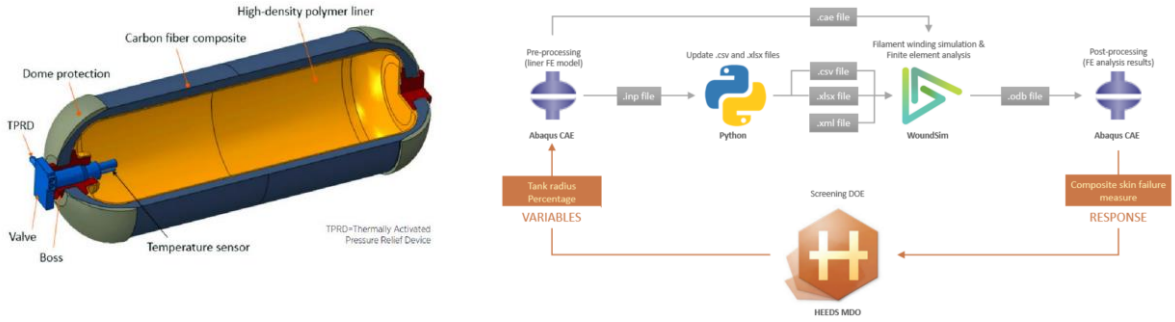
While in the aerospace, largely regarded as one of the leader sectors in applying design optimisation, are still challenged in describing a design option that can then be fed into a diverse set of models that represent an aircraft with considerable scaffolding required to connect, run and collate results (Yao, et al., 2011).

2.2. Case Studies

To complement the literature review, three case studies have been reviewed and include the design optimisation of a Carbon Fibre Pressure Vessel, Brompton Bicycle and KISPE Open-Source Satellite.

2.2.1. Design Space Exploration for a Carbon Fibre Pressure Vessel

The Digital Engineering Technology and Innovation (DETI) project performed Design Space Exploration (DSE) on a carbon composite pressure vessel (Figure 3a) (Osmiani & Brown, 2021). While successful in connecting a diverse set of models (Excel, FEA and Python) and demonstrating DSE, the project took approximately six months of an engineer’s time to connect the five models and run the DSE (Figure 3b). The time spent was attributed to the challenge of connecting models and the variety in interfaces (i.e., input and output formats). Error handling also proved problematic with experiments failing, results being lost, and challenges in debugging the set of models.



(a) Parameterised and modularised carbon fibre pressure vessel.

(b) Combined model set to perform Design Space Exploration.

Figure 2. DETI DSE demonstration.

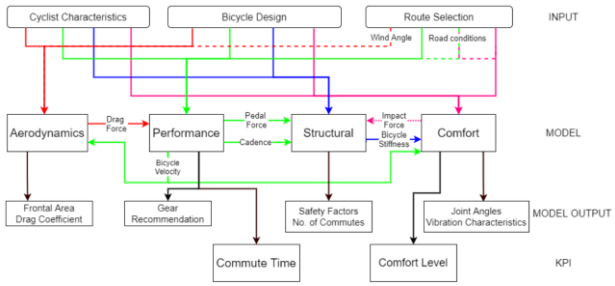
https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press

2.2.2. *Digital Twin for the Design and Redesign of a Brompton Bicycle*

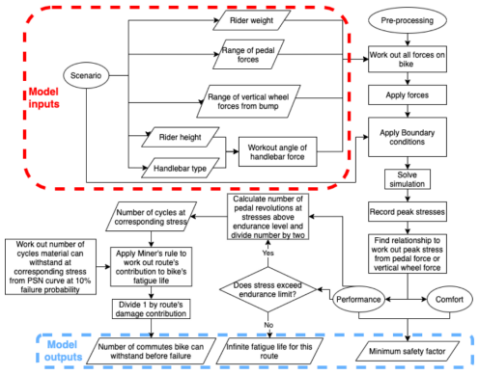
A student group project sponsored by Brompton created a Digital Twin of a bicycle and ran a design optimisation to support their continuous product improvement programmes (Figure 4a) (Diego, et al., 2021). The Digital Twin featured four numerical models (Figure 4b): a Reynolds Average Navier-Stokes (RANS) with K-Omega SST turbulence model using STAR CCM+ to evaluate aerodynamic performance (Figure 4c); a custom drivetrain performance model written in Python; Finite Element Analysis (FEA) model using Autodesk Fusion360 to evaluate structure performance (Figure 4d); and a custom comfort (multi-degree of freedom lumped mass system) model written in Python.

These models were run together to evaluate design options. The student's cost evaluation of creating the Digital Twin highlighted a capital expenditure of £ 2,500 and an operational expenditure of £ 50,000 if they had costed at market rate. The development time was four months by four individuals. Challenges in connecting the models, handling software dependencies and differences in model runtimes and resource requirements, and model version control were identified.

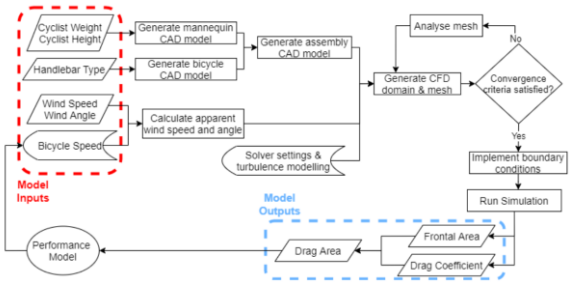
(a) Brompton Bicycle.



(b) Digital Twin for Design Optimisation and Continuous Product Improvements Programmes.



(c) Structural model workflow.



(d) Aerodynamics model workflow.

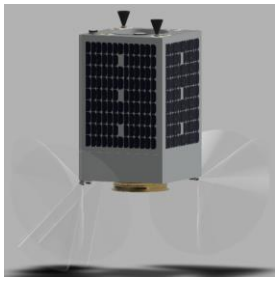
Figure 3. Student group project in partnership with Brompton.

2.2.3. *KISPE Open-Source Satellite*

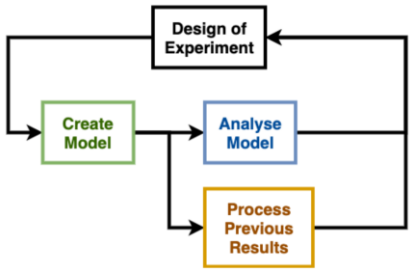
An industry partner developed a Design of Experiments (DoE) for the structural evaluation of KISPE's Open-Source Satellite (OSSAT) using commercial Finite Element Methods and an in-house Python-based DoE tool. The project was a success in developing a DoE of the structural behaviour of the OSSAT. The development time for the DoE was 354.5 hours (~15 days) with the majority of time attributed to the complexity of setting up a FEM model that could run consistently across the design space[6] and the time taken to configure the DoE. Expanding beyond structural analysis, one could assume configuring a DoE process for all the requirements of the satellite would be many months work. Error handling also played a factor in creating the DoE processes.

[6] Honeycomb structures proving difficult to model consistently when parameterised leading to errors in the model output.

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press



(a) OSSAT concept



(b) DoE Process

Figure 4. DoE for OSSAT.

2.3. Summary of the Challenges in Moving to Through-Lifecycle Whole-Design Design Optimisation

The academic reviews and case studies all highlight approaches to design optimisation are well-known however, there are many practical challenges that need addressing. These are summarised in Table 1.

Table 1. Challenges in achieving Through-Life Whole-Design Design Optimisation

#	Challenge	Source
1	The handling of a diverse set of hardware compute requirements to run the set of numerical of models.	(Yao et al., 2011) Case Study 2.2.1 & 2.2.2
2	The handling of a diverse set of software (runtime) requirements to run the set of numerical of models.	(Yao et al., 2011) Case Study 2.2.1 & 2.2.2
3	The deployment of design optimisation at varying scales of computational resource (laptop and desktop through to HPC and cloud).	(Maine & Narasinga Rao, 2017; Mala-Jetmarova at al., 2018)
4	The ability to extend to hundreds/thousands of numerical models with minimal overhead in configuration.	(Maine & Narasinga Rao, 2017)
5	The management of numerical models for traceability and auditability - version control, configuration and history of design optimisation runs - enabling the Design Optimisation to evolve along with the models throughout the product lifecycle.	(Wang et al., 2017)
6	The ability to employ and/or suggest different optimisation approaches (e.g., grid search, genetic, swarm).	(Wang et al., 2017; Mala-Jetmarova et al., 2018)
7	The cost of current solutions and licensing models are proving to be barriers.	2.2.1
8	The need to minimise configuration and set-up time.	(Yao et al., 2011) All case studies
9	The complexity of current data and information interfaces between numerical models and the optimisation process.	(Yao et al., 2011) Case Study 2.2.1 & 2.2.2
10	The management of the pre, execute and post processing of numerical models.	(Yao et al., 2011) Case Study 2.2.3
11	The management and orchestration of the entire process - which models to simulate and when, and handling exceptions, errors, and issues from models.	(Yao et al., 2011) Cast Study 2.2.1
12	The sustainable execution of the Design Optimisation (e.g., stopping the execution of numerical models if one has already returned indicating a design option is unsuitable).	(Mala-Jetmarova et al., 2018)
13	The modularisation and de-coupling numerical model creation and validation, and their inclusion into a Design Optimisation process.	Case Study 2.2.2 & 2.2.2
14	The digital skills and knowledge required to operate and configure the numerical model ensemble for Design Optimisation.	(Wang et al., 2017) Case Study 2.2.1

3. Investigating Software Deployment Toolchains as a Solution

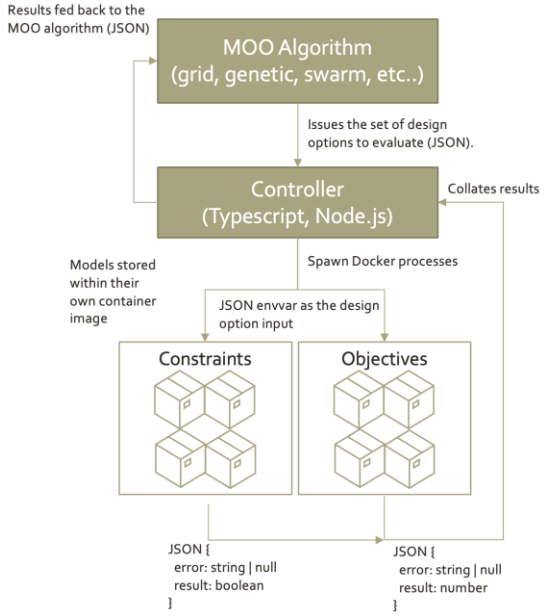
To overcome the challenges, the authors examined and trialled a solution using opensource software deployment toolchains. An action-based research approach was applied to evaluate the practicality of opensource software deployment toolchains to overcome the challenges, in particular, containerization and orchestration.

The solution is depicted in Figure 5a with 5b demonstrating the implementation. It has been called Docker Design Optimisation (DoDO) and consists of three main elements:

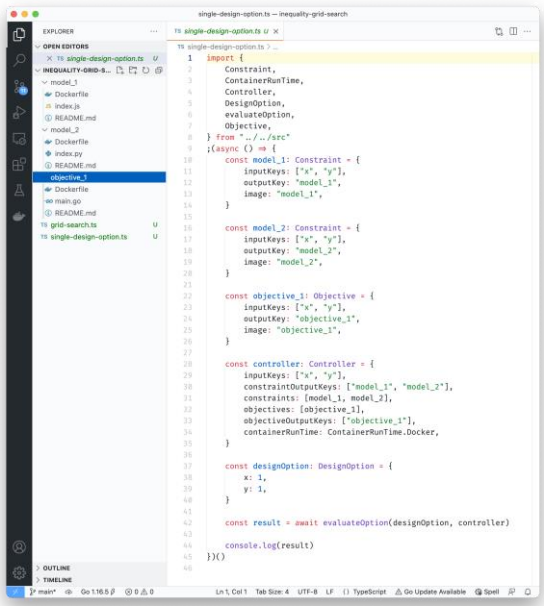
1. The selection of the design optimisation approach.
2. The creation of the *Controller*.
3. The containerisation of the numerical models that will be deployed by the *Controller*.

The first element is the selection of the design optimisation approach. Key to the solution's function is in setting a consistent interface between the approach and the *Controller*. This is achieved by describing design options in the form of JSON dictionaries as well as accepting results from numerical models in JSON.

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press



(a) Solution workflow.



(b) Code required to configure and evaluate a single design option using DoDO (written in Typescript)

Figure 5. Through-Life Whole-Design Design Optimisation using opensource software deployment toolchains - DoDO.

The second element is the *Controller*. The *Controller* listens for design options issued by the design optimisation algorithm and orchestrates the numerical model runtimes. This takes inspiration from opensource tools for automating deployment, scaling, and management of containerized applications[7]. The *Controller* is configured by detailing the numerical models that need to be run in order to evaluate a design option. The *Controller* also performs error-checking before runtime to ensure the design option description (e.g., JSON) features the input parameters required to run the numerical models.
The third element features the numerical models, which are stored as container images. Containers have the advantages of:

- storing all the dependencies required to run a model thereby avoiding conflicts between other model's requirements and providing consistent runtime behaviour. This enables all manner of model such as, CFD, FEA, ML, CAD, CAM, Excel, and General-Purpose Programming Language scripts to be run together.
- enabling model version control as the container images used to create the images can be tagged using versioning standards, such as semantic versioning.
- accepting environment variables as inputs at runtime enabling us to pass the design option variables.
- outputting the result through standard output (STDOUT) for easy collection and interpretation by the controller that can be fed back to the design optimisation algorithm to decide the next set of design options to evaluate.
- being well-supported via the Open Container Initiative (OCI) with many implementations[8] that can run on local computers through to High-Performance Computing instances thereby enabling scalability.
- resource allocation enabling optimised resource use during a design optimisation exercise.

The *Controller* manages this process, issuing commands to run containers as well as collating their results or noting failures. The failure of a container does not hinder or stop any of the other containers

[7] E.g., Kubernetes.
[8] E.g., Containerd, Docker, Singularity and Podman

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press

running thereby aiding error handling. In addition, the *Controller* presents the opportunity to stop models during their execution if another model returns a result that indicates the design option does not meet the criteria. This can reduce unnecessary resource use. In theory, the *Controller* can orchestrate hundreds to thousands of models with the limitation being the hardware it is deployed upon.

A key advantage of applying an opensource software deployment approach is the de-coupling of the design optimisation approach and the orchestration of the models, and the orchestration of the models and numerical models. This enables algorithms to be developed that need only consider setting and accepting design options and their respective scores via a JSON interface. And enables engineers to solely focus on developing and validating the numerical models that will feature in the optimisation.

In practice, the steps that need to be taken to build and run the solution are:

1. Select your design optimisation approach (e.g., grid search, genetic, gradient descent)
2. List your design option variables and set their limits (in JSON)
3. Form container images incorporating the numerical models. The image and model runtimes need to:
   a. Pre-process the JSON environment variable
   b. Initialise the model.
   c. Execute the model.
   d. Post-process of the results and print it via STDOUT for the *Controller* to access[9].
4. List the container images for the *Controller* to run during the design optimisation.
5. Run the design optimisation process[10] .

Table 2 provides a summary of how the approach meets the challenges identified in Section 2.

Table 2. Satisfying the challenges for Through-Life Whole-Design Design Optimisation

#	Challenge	Solution
1	The handling of a diverse set of hardware compute requirements to run the set of numerical of models.	Provision of computer resource can be performed at container runtime, such as number of CPUs, GPUs, RAM, and storage.
2	The handling of a diverse set of software (runtime) requirements to run the set of numerical of models.	Containers can feature Linux or Windows as their base OS with runtime requirements added to the container image. This enables them to be transferred across compute platforms.
3	The deployment of design optimisation at varying scales of computational resource (laptop and desktop through to HPC and cloud).	Container runtimes exist for laptops, desktops, cloud and HPC platforms.
4	The ability to extend to hundreds/thousands of numerical models with minimal overhead in configuration.	Software deployments exist that feature hundreds/thousands of containers running concurrently.
5	The management of numerical models for traceability and auditability - version control, configuration and history of design optimisation runs - enabling the Design Optimisation to evolve along with the models throughout the product lifecycle.	Container images can be stored in image repositories (e.g., DockerHub) that can provide version control to the numerical models. Specific versions can be requested at runtime.
6	The ability to employ and/or suggest different optimisation approaches (e.g., grid search, genetic, swarm).	The *Controller* sits between the Design Optimisation algorithm and orchestration of the models. The algorithms need only to receive and send JSON objects featuring the design options to be evaluated and collated results from evaluating a design option.
7	The cost of current solutions and licensing models are proving to be barriers.	The platforms used are opensource and free to download and use.
8	The need to minimise configuration and set-up time.	Little to no changes need to be made to run the numerical models. Time is required to create the appropriate container image to run the model.
9	The complexity of current data and information interfaces between numerical models and the optimisation process.	Interfaces between the elements of the solution are JSON.
10	The management of the pre, execute and post processing of numerical models.	Each model is managed within its own container.
11	The management and orchestration of the entire process - which models to simulate and when, and handling exceptions, errors, and issues from models.	Handled by the *Controller*.
12	The sustainable execution of the Design Optimisation (e.g., stopping the execution of numerical models if one has already returned indicating a design option is unsuitable).	Can be achieved and enabled via the *Controller*.
13	The modularisation and de-coupling numerical model creation and validation, and their inclusion into a Design Optimisation process.	Achieved through the containerisation process being the means to incorporate an existing model into the Design Optimisation process.
14	The digital skills and knowledge required to operate and configure the numerical model ensemble for Design Optimisation.	Tutorials and guides to be provided that demonstrate how to set-up and run Design Optimisation via the approach.

4. Demonstration

To demonstrate, the solution has been applied to an inequality problem one would find in design optimisation tutorials. There are two inequalities. First, the design option has to exist above a cosine curve and second, the design option has to exist below a negative gradient line (Figure 6). The

[9] In the case of constraints, this could be a Boolean result (True/False) and for objectives, it could be a Float denoting a score. In each case, an error is also included in the JSON object and is null if the model ran as expected or a string detailing the error that occurred.

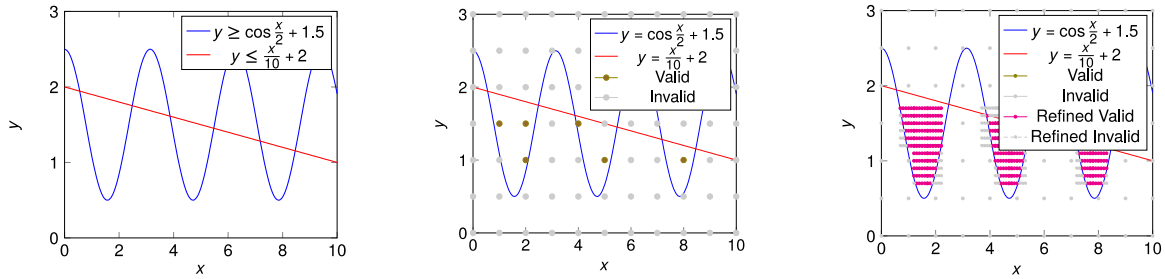
[10] Currently achieved through a command line interface.

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press

objective is to get as close to (0,0) as possible. This is scored by calculating the Euclidean distance from (0,0) to the design option's position. The problem statement is as follows:

$$\min f(x) = \sqrt{x^2 + y^2}$$
$$y \geq \cos\left(\frac{x}{2}\right) + 1.5 \quad (1)$$
$$y \leq \frac{x}{10} + 2$$

The inequalities form a disjoint design space introduced by the intersection of the two lines (Figure 6a). Each equation represents a numerical model, which were created and containerised. To demonstrate DoDO's flexibility, each model was written in a different code language (JavaScript, Python and Go). The problem was then solved via a refined grid search approach. The approach starts with a coarse grid and then performs a further, more detailed grid search about the design options that were identified as valid. The results are displayed in Figure 6 and shows the grid search performs as expected systematically evaluating design options across the design space (Figure 6b). Figure 6c then demonstrates the refined search based on the valid solutions identified in the coarse search. The creation and running of the optimisation took less than an hour with most of the time spent creating the models rather than instantiating them within the design optimisation process.



(a) Inequality optimisation problem (b) Single Grid Search (b) Refined Grid Search (n=2)

Figure 6. DoDO results for the inequality design problem.

5. Discussion and Future Work

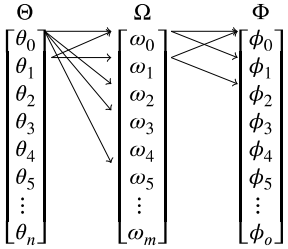
The demonstration evidences the potential of using opensource software deployment toolchains to develop and solve design optimisation problems. Table 2 also shows how software deployment toolchain approaches can overcome the challenges of Through-Lifecycle Whole-Design Optimisation. However, further development is required to make the approach robust and industry ready. First, is the validation of data going from and to the three elements of the approach and for the design optimisation algorithms to accept and/or consider any errors from the models being run. The area features a wealth of tools and standards that could be employed. Examples include JSON validation via JSON schema, and standards, such as Simulation Model Portability (SMP).

Second, is the need for case studies to evaluate the time to develop, deploy and run design optimisations in order to determine the potential Return on Investment (RoI). This also needs to consider the complexity of the design problem and set of numerical models used to evaluate the design space. In addition, means by which one can describe the nature of a design space at an early conceptual level could support organisations in deciding which, if any, design optimisation approach is necessary.

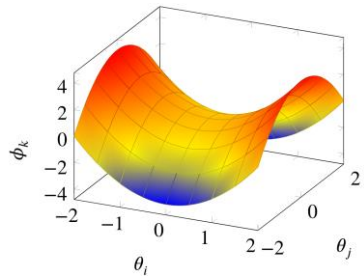
Third, is an opportunity that the approach affords due to its ability to co-ordinate model runtimes and the potential to not have to evaluate a design option in its entirety. This could be useful where some models may be computationally expensive to run, and one might want to exit them early if they are aware that a design option has failed some other criteria. This can be mathematically described by saying we have n parameters, θ, that we use to produce a design. Therefore, a unique combination of specified design parameters forms a design option, Θ, $\Theta = [\theta_1, \ldots, \theta_n]$. Aggregating design options gives us the design space, D, $\mathrm{D} = [\Theta_1, \ldots, \Theta_\mathrm{N}]$. We then want to assess the performance of the design

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press

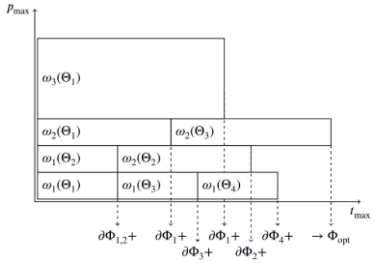
options through m parameters, ϕ. Thus, we will have a design option performance assessment, Φ, $\Phi = [\phi_1, \ldots, \phi_m]$ for each Θ. Combined, they form the assessment space, A, $A = [\Phi_1, \ldots, \Phi_M]$.
We then use our numerical models to map a design option to a performance assessment, $\Theta_i \rightarrow \Phi_i$. Fundamentally, a numerical model, ω, can be represented as a function of Θ, $\omega = f(\Theta)$, and uses Θ, in part or in full, to generate some information on Φ (Figure 7a). In fact, it is possible that multiple subsets of our model ensemble will be able to provide the information required to fully describe a design option Φ, $\Omega: \Theta \rightarrow \Phi$, albeit at different fidelities. Equally, there may be some models that can only validate portions of the design space that one wishes to explore limiting the detail and increasing the level of uncertainty across areas of the design space (Figure 7b).



(a) A design option, Θ, being used by our set of models, Ω, to provide us with the performance assessment, Φ.



(b) A model, ω, that is able to evaluate a design sub-space for ϕ_k based on a subset of values in Θ.



(c) A set of functions configured to run in time, t, and level of parallelism, p, permitted. Each function contributes to our understanding of ϕ *b*ut how do we do this efficiently?

Figure 7. Evaluating a design option through our ensemble of models.

Models also require some resource to perform their function. In our problem, we define two types of resource. The first is the number of concurrent processes[11], p, and the second is the time it will take to run, t. The two dimensions are important as an organisation will be both parallel compute, p_{max}, and time, t_{max}, bound. Thus, we define total resource used as a function of processes and time, $R = f(p, t)$.
With the scene set, the research question is: Given some requirement for Φ, we wish to determine either an optimum Θ or a subset of Θ that satisfies our requirement in the most resource efficient manner possible, $\Rightarrow \min R$. Equally, given fixed t and p and the desire to map and/or describe the design space, what set of models and design options should one evaluate to give the most information about the design space? Interesting features of this problem includes models reporting $\partial\Phi$ and require varying levels of p and t, and that we have an array of models at our disposal of varying fidelity (Figure 7c).
The hypothesis is that there are configurations that provide an optimal search of the design space using the partial results attained during run-time to determine what models should be run next and with what Θ. Also, long computations could be halted if new information arises resulting in the computation no longer being required. Equally, there are decisions to be made with regards to starting a model based on the current information available or waiting for additional information to be returned from other models. The final aspect is in determining what models we should develop for our design problem. In essence, this problem brings the fields of scheduling, numerical design of experiments, design exploration and design optimisation together to provide a systems-level overview of our design.
Finally, it is acknowledged that there is an extant set of industry tools that are seeking to address Through-Life Whole-Design Design Optimisation and the research team are currently reviewing these toolsets with respect to the requirements elicited to assess whether and how they meet these requirements. The objective is to reveal what the misconceptions and barriers are in using the existing toolsets, and how we might bridge this gap so that Through-Life Whole-Design Design Optimisation can be adopted.

[11] Typically limited by compute resource.

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press

6. Conclusion

Through-Life Whole-Design Design Optimisation is considered one of the next steps in design. The desire to apply Through-Life Whole-Design Design Optimisation exists as well as the need with product requirements increasing to include factors brought about by the Circular Economy, sustainability, and decarbonisation. However, the varied nature of our modelling capability from low to high-fidelity, low to high computational complexity, distinct runtimes and compute environments, and the considerable time taken to interface between models has made it challenging and resource intensive to apply existing implementations of design optimisation. This is leaving industry to question its Return-on-Investment. This paper identified 14 underlying challenges preventing Through-Life Whole-Design Design Optimisation adoption via a review of review papers and recent design optimisation case studies.

This paper has also demonstrated how opensource software deployment toolchains could be exploited to enable Through-Life Whole-Design Design Optimisation due to their ease of set-up, ability to run at any scale and across models featuring unique compute requirements. Through-Life can be achieved as software deployment toolchains can readily accommodate updates to models, the inclusion of new models and deprecation of old models. Equally, the version control afforded by these toolchains enables business to 'revisit' any point of the lifecycle so they can evaluate how the design evolved over time. Whole-Design can be achieved through containerization that enables numerical models to manage and run concurrently without fear of conflicts. These capabilities align with Model-Based Systems Engineering and Set-Based Design paradigms. Software deployment toolchains as a means to orchestrate and manage design optimisation could pave a new wave of adoption across the industry. In addition, the ability to partially evaluate design options opens up opportunities to explore the optimal evaluation of a design space given a fixed amount of compute and time resource, as well as enable us to explore what to model, to what fidelity and when in the lifecycle.

Acknowledgements

The work has been undertaken as part of the Engineering and Physical Sciences Research Council (EPSRC) grants – EP/R032696/1 and EP/V05113X/1 – and Digital Engineering and Technology & Innovation (DETI) project.

References

Crowley, D., 2013. An efficient approach for high-fidelity modeling incorporating contour-based sampling and uncertainty, s.l.: s.n.

Diego, G., Theodoli, C., Podmore, L. & Tejuoso, D., 2021. Investigating the role of a digital twin in the design and redesign of a Brompton bicycle, Bristol: Group Industrial Project (GIP) - University of Bristol.

Handawi, K. et al., 2021. Scalable Set-Based Design Optimization and Remanufacturing for Meeting Changing Requirements. Journal of Mechanical Design, 143(3).

LaSorda, M., Borky, J. & Sega, R., 2018. Model-based architecture and programmatic optimization for satellite system-of-systems architectures. The Journal of The International Council on Systems Engineering, 21(4), pp. 372-387.

Mala-Jetmarova, H., Sultanova, N. & Savic, D., 2018. Lost in Optimisation of Water Distribution Systems? A Literature Review of System Design. Water.

Mane, S. & Narasinga Rao, M., 2017. Many-Objective Optimization: Problems and Evolutionary Algorithms - A Short Review. International Journal of Applied Engineering Research, pp. 9774-9793.

Osmiani, C. & Brown, A., 2021. Report on Design Space Exploration, Bristol: Digital Engineering Technology and Innovation (DETI) project.

Songqing, S. & Wang, G., 2004. Space exploration and global optimization for computationally intensive design problems: a rough set based approach. Structure and Mulitdisciplinary Optimization, 28(6), pp. 427-441.

Stump, G. et al., 2007. Visual sterring commands for trade space exploration: User-guided sampling with example. s.l., International Design Engineering Technical Conference and Computers and Information in Engineering Conference.

Wang, H., Olhofer, M. & Jin, Y., 2017. A mini-review on preference modeling and articulation in multi-objective optimisatio: current status and challenges. Journal of Complex Intelligent Systems, pp. 233-245.

Yao, W. et al., 2011. Review of uncertainty-based multidisciplinary design optimization methods for aerosapce vehicles. Progress in Aerospace Sciences, pp. 450-479.

https://doi.org/10.1017/pds.2022.188 Published online by Cambridge University Press