DEPLOYING HEALTHCARE SIMULATION MODELS USING CONTAINERIZATION AND CONTINUOUS INTEGRATION

Alison Harper a.l.harper@exeter.ac.uk 0000-0001-5274-5037
Thomas Monks t.m.w.monks@exeter.ac.uk 0000-0003-2631-4481
Sean Manzi s.s.manzi@exeter.ac.uk 0000-0003-2631-4481

PenCHORD, PenARC
University of Exeter Medical School
St Luke's Campus, Heavitree Road

Keywords: simulation, deployment, containerization, continuous integration, healthcare, open source

ABSTRACT

Methods or approaches from disciplines outside of OR Modeling and Simulation (M&S) can potentially increase the functionality of simulation models. In healthcare research, where simulation models are commonly used, we see few applications of models that can easily be deployed by other researchers or by healthcare stakeholders. Models are treated as disposable artifacts, developed to deliver a set of results for stakeholders or for publication. By utilising approaches from software engineering, M&S researchers can develop models that are intended to be deployed for re-use. We propose one potential solution to deploying free and open source simulations using containerisation with continuous integration. A container provides a self-contained environment that encapsulates the model and all its required dependencies including the operating system, software, and packages. This overcomes a significant barrier to sharing models developed in open source software, which is dependency management. Isolating the environment in a container ensures that the simulation model behaves the same way across different computing environments. It also means that other users can interact with the model without installing software and packages, supporting both use and re-use, and reproducibility of results. We illustrate the approach using a model developed for orthopaedic elective recovery planning, developed with a user-friendly interface in Python, including a clear set of steps to support M&S researchers to deploy their own models using our hybrid framework.

1 INTRODUCTION

Hybrid modeling in the field of Modeling and Simulation (M&S) refers to the combined use of simulation techniques with frameworks, methods, tools, and approaches from diverse disciplines. In addition to hybrid simulation, which mainly focuses on model implementation, hybrid modeling leverages knowledge artifacts from fields such as Software Engineering and Applied Computing (Mustafee et al. 2020). Combining methods from these fields can enhance the functionality or usability of simulation models.

In healthcare, simulation is used extensively to understand, analyse, and optimise complex systems, contributing to enhanced efficiency, effectiveness, and preparedness (e.g. Philip et al. 2022; Salmon et al. 2018). Frequently, complex models are developed for a single problem in a single organisation, results are generated, a report is produced, and the study is published. This means that decision-makers no longer have access to the model should they wish to investigate additional scenarios. Further, across health and care services, similar problems are seen in similar service configurations. These situations contribute to model waste, as models are not effectively integrated into decision-making processes, and subsequently, duplicated effort is required as researchers continue to develop similar models for similar problems.

A further potential issue with simulation modelling is reproducibility of simulation results. The ability to reproduce published results is central to the scientific method. The reproducible research movement in data science and M&S is encouraging researchers to make their code or model available, along with a

description of the hardware and software environments (Monks et al. 2019; Auer et al. 2021), but even this level of detail may not be enough to replicate results (Moreau, Wiebels & Boettiger, 2023).

Developing models to be deployable by healthcare staff or for results replication ensures that they are ready for use in a specific operational environment. However, there are significant challenges to deploying models for these purposes. The use of free and open-source software overcomes cost barriers associated with license fees but can create new challenges. For example, simulation models require software to be downloaded and installed, integration with existing systems, computational resources, and model maintenance and updates. For reproducing results, the correct version of the operating system, software, packages, allocation of memory resources etc. may be required. While formal dependency management tools can be used, further problems can occur when a project or application relies on many dependencies, and these dependencies have complex and conflicting requirements or dependencies of their own. This is further complicated when the model needs to be deployed across different operating systems, for example across both Windows and MacOS.

Deploying models through virtualisation overcomes a number of these barriers (Byrne et al. 2010). One way of achieving this is through a hybrid application of simulation with containerisation and continuous integration. Continuous integration involves frequently merging code changes – which may be from multiple collaborating modellers – into a central repository. Modellers can use version control systems, such as Git, to manage their code changes. These can be done in development branches, which can be merged into the main branch as work on a specific feature or bug fix is completed, and once tests or other quality checks are passed. Once the build and tests pass successfully, a new release can be created and made available for deployment via containerisation.

Containers provide a self-contained environment that encapsulates the model, and all the dependencies required by it. This isolation ensures that the dependencies within the container do not conflict with the host system or other applications running on it. We illustrate the approach using a free and open source discrete-event simulation (DES) model developed for orthopaedic elective COVID-19 recovery planning, including a clear set of steps to support M&S researchers using our hybrid deployment framework (introduced in Section 4).

2 DEPLOYMENT CONCEPTS

2.1 Free and Open-Source Software (FOSS)

Free and Open-Source Software (FOSS) tools are software applications or programs released under a license that allows users to freely use, modify, and distribute the software's source code (Dagkakis & Heavey, 2016). While containerisation does not inherently rely on FOSS, many containerisation solutions such as Docker are built on top of FOSS technologies, and its core components are open source. Typically a FOSS operating system is based on a customisation of Linux. The licenses used with Linux grant users the freedom to redistribute (possibly adapted) copies under the same terms and conditions, it can be used for any form of computing, and no license cost is involved. FOSS tools are often characterized by transparency, community-driven development, and the ability for users to customize and adapt the software to their specific needs. Before sharing code, it is essential that authors select an appropriate FOSS license for the code and other research artefacts to ensure appropriate use of the model and remove liability of the authors. A common approach in data science is to adopt a permissive license such as MIT (see Monks et al. (2022) for more details on license types).

Python is a popular FOSS programming language known for its simplicity and versatility. Python discrete-event simulation (DES) packages include SimPy (Team SimPy 2020), Salabim (van der Ham, 2018), Ciw (Palmer et al. 2019), and de-sim (Goldberg and Karr 2020). These have been used in several Operations Research relevant publications in healthcare (e.g. Allen et al. 2020; Chalk et al. 2021; Mohd et al. 2021; Anagnostou et al. 2022). For agent-based modelling, Python libraries include Mesa (Kazil et al. 2020) and for System Dynamics models, PySD (Martin-Martinez et al. 2022). PySD is also available in R via the PySD2R package, while DES simulation libraries available for R include R Simmer (Ucar et al.,

2019) and for Julia, SimJulia (Lauwens, 2021). Packages also exist for providing user-friendly, interactive front-ends to open-source models. These include Streamlit for Python (Streamlit, 2023) and Shiny for R (Shiny, 2023). These are browser-based technologies that can be deployed on the web using (for example) streamlit.io or shinyapps.io, although these come with resource limits and may not handle high user load.

2.2 Containerisation tools

In computing, *containers* refer to lightweight, portable, self-contained software units that package together an application, such as a simulation model, and all its software dependencies, including libraries, frameworks, data, and system tools. Containers provide a consistent and isolated runtime environment for running applications, ensuring that they behave the same way across different computing environments (Merkel, 2014; Shasha et al. 2022). Each container operates as a separate entity, with its own file system, networking, and resources, while sharing the host operating system's kernel. Containers enable efficient deployment, scaling, and management of applications, as they can be easily moved between different computing environments without compatibility issues.

Commonly used open-source tools for developing, managing and running containers include Docker (Merkel, 2014), Singularity/Apptainer (Kurtzer et al. 2017), and Podman (Podman, 2023). As well as encapsulating all dependencies within the container to avoid conflicts, containers support reproducibility, as the exact same environment is used across systems (Krafczyk et al. 2019).

A *container registry* is a centralised repository used to store and distribute container images. An *image* is a blueprint for what you want to build, including the operating system, software and packages. A container is an instantiation of an image: multiple copies of the same image can run simultaneously. When a container image is built, it is pushed to a container registry, where it can be pulled by different teams, systems or cloud platforms. Popular container registries include DockerHub, Google Container Registry, and Microsoft Azure. These enable efficient collaboration and deployment in container-based environments. The container is portable, in that once created, it can run on any system that supports containerisation technology without modification, and supports scalability as the application can be easily replicated and deployed across multiple instances. It provides an excellent solution for deploying interactive applications such as Streamlit and Shiny. While sharing dependencies may be an appropriate approach, wrapping everything in a container such as Docker and posting it to a public (or private) container registry such as DockerHub, reduces the need to recreate the environment and makes the model more accessible for users.

2.3 Code repositories

Central to deployment using containerisation is a remote code repository. The most popular solutions are GitHub, GitLab, and BitBucket. All code artefacts (including licence, meta-data files, and README) should be committed to the remote repository to provide long-lasting version control.

GitHub and other online code repository's primary purpose is version control of code. That is, modellers incrementally commit updates to their code and models during development. These tools allow for easy *rollback*, inspection, or use of older versions of the code. Another major feature of version control is *branching*. The main (sometimes called master) branch of a repository contains the *production model* I.e., a simple, but functional simulation model that can be deployed to users. Incremental developments are typically made on a development or feature branch, tested and then merged into the main branch when a modeller is confident it will not break the production model. GitHub is an online tool. In traditional workflows, all of the model coding is done offline in a local code repository that is managed by a local version control system such as *Git*. A modeller codes their model on their own machine, and *pushes* their commits to GitHub (or similar). A new user of the code would *clone* the GitHub repository to their own local machine. A user who is already using the code might *pull* updates from the remote repository and merge changes into their own branch.

As a simple example, consider a SimPy model of an emergency department (ED). The model logic is

fully coded and it can be run from the command line. The working SimPy model is currently committed to the main branch. An NHS analyst can clone the main branch of the repository and execute the model to analyse their ED. Simultaneously, a simulation modeller can develop a graphical interface for the model on a separate feature branch (e.g. called *interface*). When the interface is complete the modeller can issue a pull request to the main branch. A pull request is effectively a request to merge code from one branch into another e.g. from *interface* to main. It provides a safety net for the changes to be reviewed before merging (and potentially breaking a production model). Once merged the NHS analyst can simply pull the new code from the main branch, although there are no guarantees the new model interface will install easily on the NHS analyst's machine.

A secondary use of tools like GitHub is for collaboration on scientific studies and model development. GitHub provides many mechanisms for its users to share, review, and control code. One simple example is *Issues*. This is a discussion log that might report bugs, potential improvement, new ideas, and general queries. Another example is the concept of a *release*. A release tags a snapshot of the code. In our ED model example, the SimPy command line model might be tagged as version 1.0.0. While the ED model with a graphical user interface might be tagged as version 2.0.0. A release is an easy way for modellers to talk to their users about the correct version of the model: it supports a shared language between modellers and users.

2.1 Continuous Integration and Continuous Deployment

Continuous practices include continuous integration (CI) and continuous deployment (CD). These are software development industry practices that enable frequent and reliable releases of new features and products (Shahin et al. 2017). Whilst CI advocates integrating work-in-progress multiple times per day to a shared repository, CD is the ability to make releases quickly and reliably through automation support.

CI involves regularly integrating code changes from multiple developers into a shared code repository such as GitHub. The main goal of CI is to detect and address integration issues early by automatically building, testing, and validating the code changes.

An example is GitHub Actions, a feature provided by GitHub, enabling automation of various tasks and workflows within a GitHub repository, including CI/CD. With GitHub Actions, custom workflows can be defined using YML (Yet another Markup Language) files that describe the steps to be executed when specific events occur, such as push events (code changes) or pull request events. These workflows can include building and testing code, running linters (which flag syntax errors), performing code reviews, deploying applications, and more (GitHub, 2023).

The next section briefly describes our case study DES model and its user-interface, before we step through the stages required to build a container for the model.

3 CASE STUDY: ELECTIVE ORTHOPAEDIC CAPACITY PLANNING MODEL

3.1 Background

This case study describes an application of DES for use as a generic planning tool for regional post-COVID-19 orthopaedic waiting list recovery. Elective joint replacement surgeries are among the most common elective procedures. In the aftermath of the pandemic, elective waiting lists were at record levels in the UK. To address this challenge, the government allocated funds for healthcare organisations to expand capacity and meet new interim elective targets, with rapid planning occurring on a large scale across the UK.

The DES model has been described in detail elsewhere (Harper et al. 2023). It enables various configurations of bed numbers, theatre capacity, and productivity measures such as theatre scheduling, hospital lengths-of-stay and rates of delayed discharges to estimate resultant surgical throughput and capacity requirements, developed for a proposed orthopaedic facility. It is developed to be used by

healthcare planners, managers, and clinicians, can be reused for similar applications, and can be readily adapted for other specialties or scaled for multiple specialties.

3.2 Overview of the DES model

The FOSS DES model was developed in Python 3.8 using SimPy, with Streamlit to create a user-interface, in collaboration with a large NHS Trust in England.

In the model, patients are categorised by surgical procedure and enter deterministically according to a baseline operating theatre schedule. If a bed is not available within a tolerance of 0.5-1 day, the theatre slot is lost. A conceptual view of the model is in Figure 1. Lengths-of-stay are sampled by surgical type, and a proportion of patients have their discharge delayed, often for downstream reasons such as lack of availability of community health or social care. A scenario table is easily created for running the simulation; the model will run with the baseline theatre schedule, five days per week, with pre-defined surgical allocations.

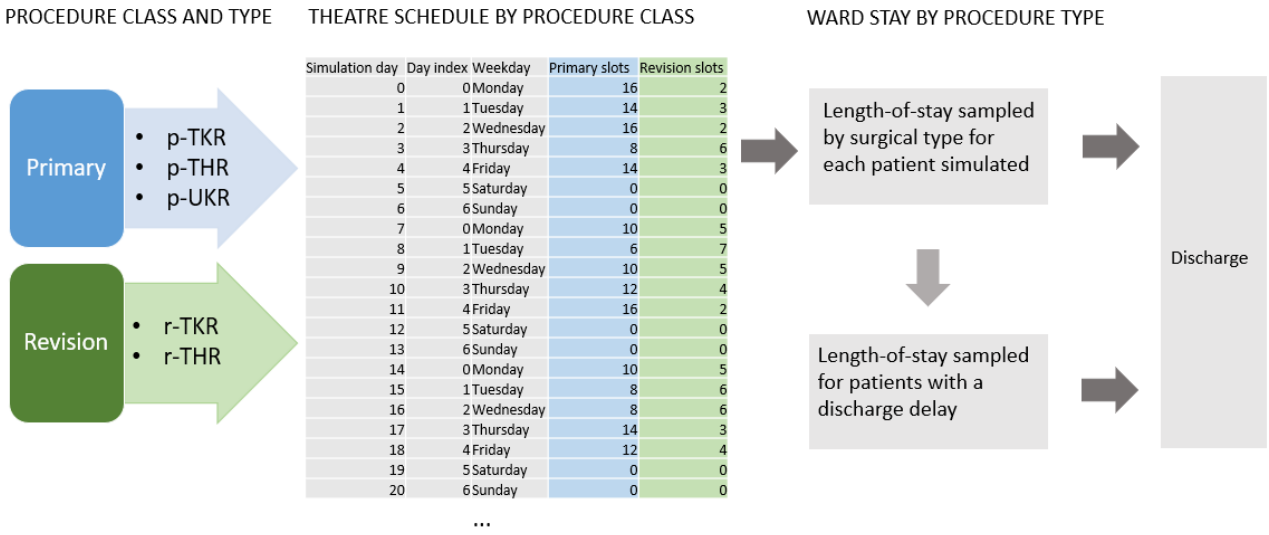

Figure 1: Conceptual process of orthopaedic model.

An optional user-defined schedule can vary, per day of week, the numbers of theatres available, the daily number of theatre sessions, and the allocation of procedure types to each session. Parameters can be changed in the Streamlit interface using sliders and buttons; these additionally include numbers of beds, patient mean lengths-of stay per procedure type, length-of-stay of patients with a delayed discharge, and proportion of patients with a delayed discharge (Figure 2). This enables comparing scenarios including resource configurations, adding evening or weekend theatre activity, and changing the way surgical procedures are scheduled, for example scheduling complex surgeries with longer, more variable lengths-of-stay earlier in the week.

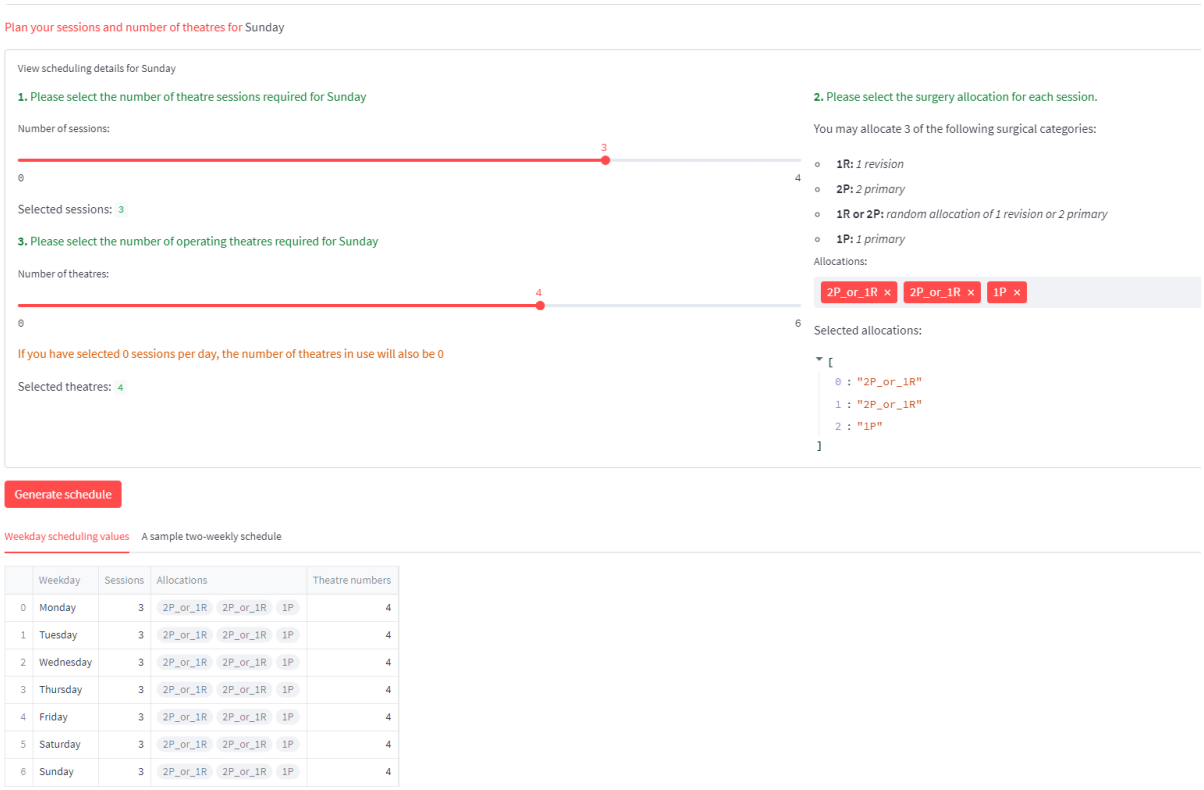

Figure 2: Setting up a user-defined operating theatre schedule by procedure class.

Should a user-defined theatre schedule be used in the model, the interface will display both the user-defined schedule rules and the baseline schedule rules side-by-side (Figure 3). A new scenario table will be generated for the previously created scenarios using both the baseline theatre schedule, and the user-defined schedule.

The model outputs include: (i) *Total surgical throughput* with the aim of identifying the configuration which best achieves this within other constraints; (ii) *Bed utilisation* to identify days of the week where the system is under pressure; (iii) *Lost theatre slots for system reasons*, which represents a mismatch between the balance of bed utilisation to theatre activities per weekday, given model parameters.

Both resourcing and productivity measures are inputs into the model. Reducing elective lengths-of-stay and delayed discharges are national priorities (Wall et al. 2022). This means that healthcare planners, managers, and clinicians are required to work together to determine the most likely combinations of parameters, accounting for other system components such as workforce. The model can be used as a planning tool, enabling discussion, and generating immediate results which can be used to provide quantitative evidence in support of a business case to secure allocated funds.

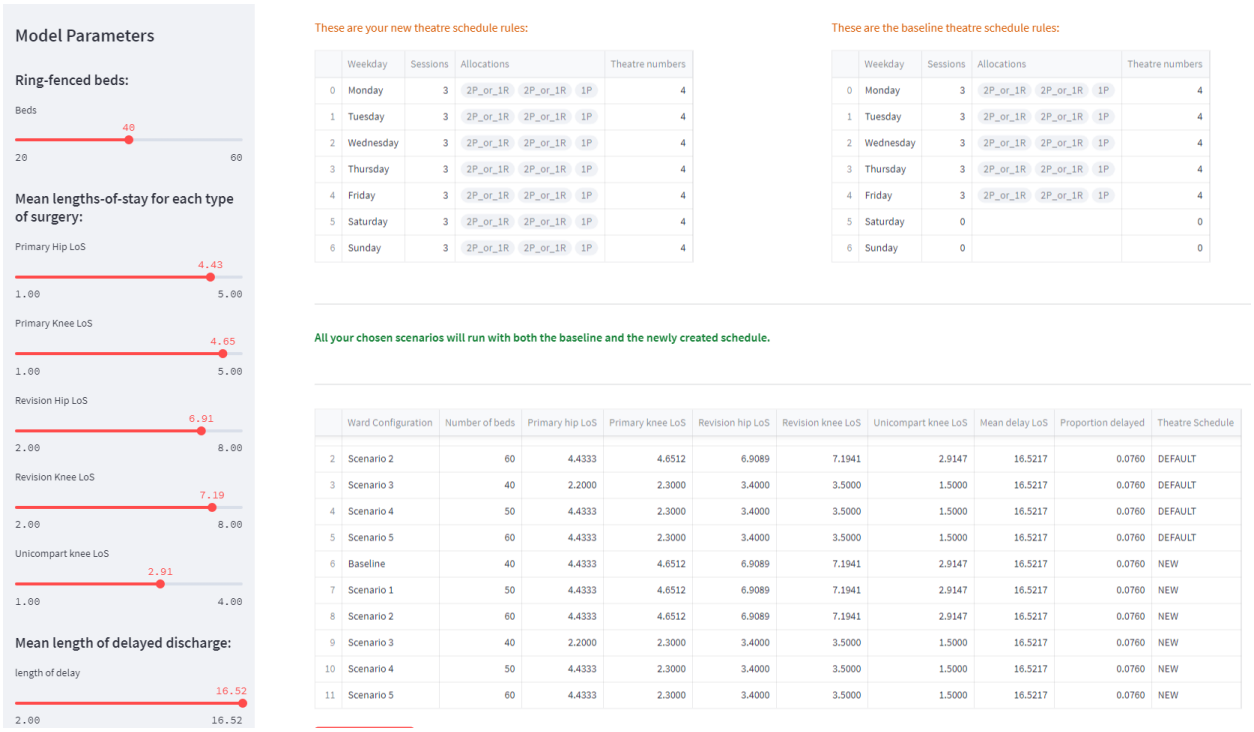

Figure 3: User-defined and baseline scenarios used for scenario table. The baseline and user-defined scheduling rules are displayed at the top, the scenario table is displayed at the bottom, with the far-right column indicating that each scenario will run with each theatre schedule.

3.3 Sample results

A sample set of results is provided in Figure 4, with illustrative means across all replications and model runs for a set of 40 parameter configurations. Bed numbers from 30 to 70 (in increments of 10, a total of 5 parameters) were investigated with the baseline theatre schedule (5 days), and a second user-defined schedule which includes weekend working (7 days), with all other theatre parameters remaining the same.

For each of these 10 parameter combinations, an additional four scenarios used all combinations of: baseline length-of-stay values, 50% of baseline length-of-stay values, baseline delayed discharge rates, and 10% of baseline delayed discharge values (summarised in Table 1). Results are in Figure 4.

Table 1: Summary of example simulation scenarios

Scenario	Length-of-stay	Delayed discharges	Beds numbers	Theatres schedule
1	Baseline	Baseline	30 – 70 (intervals of 10)	Baseline (5-day working) Baseline + weekend (7-day working)
2	50% Baseline	10% Baseline	30 – 70 (intervals of 10)	Baseline (5-day working) Baseline + weekend (7-day working)
3	Baseline	10% Baseline	30 – 70 (intervals of 10)	Baseline (5-day working) Baseline + weekend (7-day working)
4	50% Baseline	Baseline	30 – 70 (intervals of 10)	Baseline (5-day working) Baseline + weekend (7-day working)

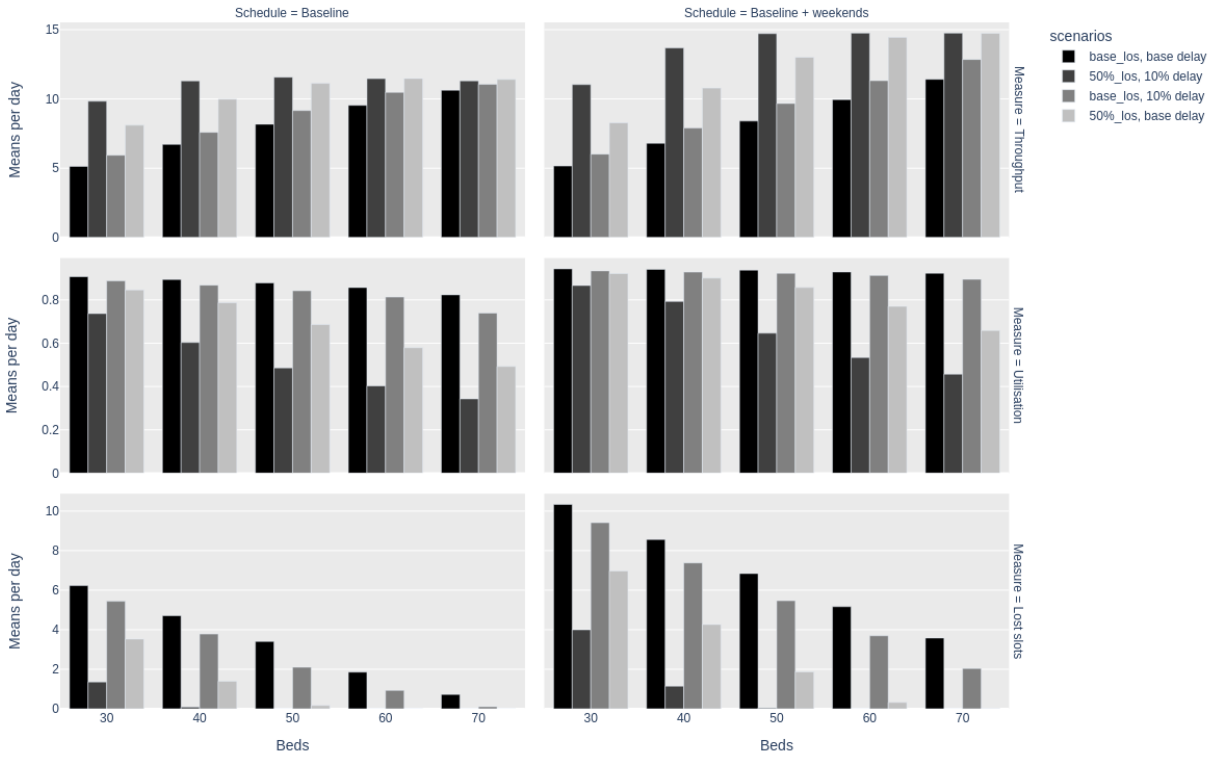

Figure 4: Example results (means for each scenario). Columns are theatre schedules, rows are output metrics. Bed numbers are shown on the x-axes. Scenarios vary length-of-stay (los) and proportion of patients experiencing a delayed discharge (delay).

Clinicians and managers can work together to investigate scenarios to gain realistic expectations of required bed and theatres numbers to maximise surgical throughput.

We show one way that this can be achieved through hybridizing the simulation model for deployment using containerisation and continuous integration and deployment. As this approach is novel in applied healthcare M&S research, we have additionally outlined the steps taken to achieve the hybrid integration. The purpose of this is to support M&S researchers interested in deploying applied healthcare simulation models to be used by NHS staff, with the aim of increasing the potential to make a real impact on healthcare service delivery.

4 DEPLOYED IMPLEMENTATION

We illustrate our deployment method using Docker for containerisation, DockerHub for container hosting, and GitHub actions for continuous integration and deployment. All simulation code, build scripts and tests are available on GitHub GitHub - Model of orthopaedic ward. and archived at Zenodo (Harper and Monks 2023, TBA). The Docker container can be found at https://hub.docker.com/r/tommonks01/hep-sim. Figure 5 illustrates our example deployment framework. Each component is explained in steps.

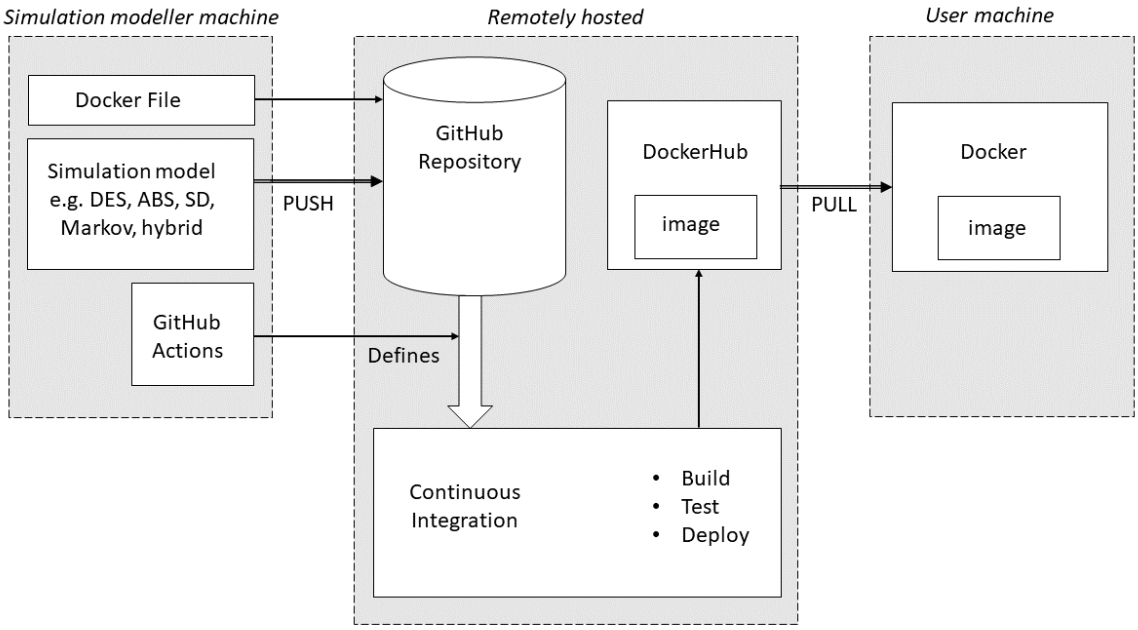

Figure 5: Hybrid deployment framework

4.1 The Docker container

The build for a Docker container is specified within a Dockerfile. Listing 1 details the Dockerfile used within our example. It is comprised of the following major steps.

Step 1: The initial image

Docker works in a modular way by building a new image based on other published images. Here we start by using *Python 3.8-slim.* This provides a light weight (smaller size) Debian Linux image with Python 3.8 pre-installed. An alternative would be to use a full version of Linux such as *Ubuntu:latest.*

Step 2: Install any extra software needed

The slim build of Linux and Python is useful to keep an image to a relatively small size. The downside is that it may miss some essential low-level software needed to run a simulation model, and Streamlit web apps. It will also exclude version control software such as Git that that we need to use to clone the GitHub repository during the build. Our second stage installs these essential tools including Git using a Linux package management tool. Any cached files are removed at the end of the installation.

Step 3: Clone the remote code repository

We intend this script to be run remotely as part of our continuous integration and deployment pipeline. We therefore use *git clone* to copy the most recent version of the code and software dependency list to our working directory on the image.

Step 4: Install the simulation model dependencies

In order for the simulation model to run we need to install the correct software dependencies (e.g. SimPy and Streamlit). In our example, we are installing our dependencies with the Python Package Index (PyPI). We are using the pip package manager for Python to install our dependencies on our image. These are held in the *requirements.txt* file stored in our git repository. You can think of this as a simple shopping list. It is a list of package names and versions.

Step 5: Create a container entry point

The last step is included to immediately launch the Streamlit front end to our simulation model when the container is run. Streamlit provide examples and instructions online. In simple terms we are opening port 8501 as this is the port used by the Streamlit web server. We then specify that the container will run Streamlit and point it at the Python module Hospital_Efficiency_Project.py. This contains our Streamlit script.

```
# Step 1: the initial image
FROM python:3.8-slim

# streamlit working dir is /app
RUN mkdir /app
WORKDIR /app

# Step 2: Install extra software
# Git means we can clone from a repo instead of local copy if needed.
# Clean up after.
RUN apt-get update && apt-get install -y \
build-essential \
git \
software-properties-common \
&& rm -rf /var/lib/apt/lists/*

# Step 3: clone the repository to the app directory
RUN git clone https://github.com/TomMonks/hep-deploy /app

# Step 4: pip install the python software dependencies e.g. simpy 4.0
RUN pip3 install -r requirements.txt

# Step 4: Container EntryPoint
# EXPOSE PORT 8501 - standard for streamlit.
EXPOSE 8501

# run app on 0.0.0.0:8501
# This means that the app will run when the image is launched.
```

```
ENTRYPOINT ["streamlit", "run", "Hospital_Efficiency_Project_Home.py", "--
server.port=8501", "--server.address=0.0.0.0"]
```

Listing 1: script used to build Dockerfile for model and streamlit app

4.2 Continuous integration of new code

Continuous integration allows us to quickly update our simulation models and their interfaces. An important part of this process is testing that our changes do not break the simulation model. The full details of testing simulation models are out of scope for this paper. Here we illustrate testing by embedding a simple test that checks the simulation model will run and return results after an update is made. We do this using two automation tools: *pytest* and GitHub actions.

Testing the model still runs

In our repository we have created a sub directory called *tests/*. The directory contains a Python model with a single python function called *test_model_can_run*(). The function simply creates an instance of the model and performs a single run. A test is successful if it returns results with non-zero length. Prefixing the name of this function means that it can automatically be discovered by the python package *pytest.* This package will run all tests it finds and returns the results to the user.

Automating pytest on new code commits

Our simple test will execute automatically each time a change is made to the code and committed to the main branch of the GitHub repository. Github actions are written in Yet another Markup Language (YML). Listing 2 details the code used in test_model.yml

In summary, listing 2 will be run if new code is pushed to the main branch of the GitHub repository or if a pull request is created (i.e. a suggested update to the main branch that is checked before merging). The tests will be run on Ubuntu Linux using Python versions 3.8 and 3.9. For each version of Python, pip is used to install the dependencies and pytest is called to automatically discover and execute tests. Users can navigate to the GitHub 'actions' tab to view all workflows and browse the results of the tests. If an error has occurred, the pull request can be rejected.

```
name: Test model

on:
 push:
branches: [ main ]
 pull_request:
branches: [ main ]

jobs:
 build:

runs-on: ubuntu-latest
strategy:
 matrix:
  python-version: ["3.8", "3.9"]
```

```
steps:
 - uses: actions/checkout@v3
 - name: Set up Python ${{ matrix.python-version }}
  uses: actions/setup-python@v4
  with:
   python-version: ${{ matrix.python-version }}
 - name: Install dependencies
  run: |
   python -m pip install --upgrade pip
   pip install pytest
   if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
 - name: Test with pytest
  run: |
   pytest
```

Listing 2: GitHub action to automate testing of code during continuous integration

4.3 Automated build and deployment to DockerHub

Our final step builds the Docker image remotely and pushes to DockerHub. We again use GitHub actions to automate this process. We aim to minimise the number of Docker builds we produce. Therefore, we limit this operation to when we issue a GitHub release. A release is when we decide that a major, minor or patch (bug fix) has been made to our simulation model code. A release might have a name or more usefully a clear version number. For example, version 0.1.1. Listing 3 details the YML instructions.

Listing 3 can be used as a template for other implementations of our continuous integration and deployment pipeline. There are two key aspects the user must adapt: credentials and the image name. Users must first create two GitHub repository secrets *DOCKER_USERNAME* and *DOCKER_PASSWORD*. These are the account details for DockerHub. They are entered via GitHub settings and are stored in an encrypted format.

Listing 3 specifies the image used in our applied example. When adapted, users must specify their own image name. This is in the format *my-docker-hub-namespace/my-docker-hub-repository*.

```
name: Publish Docker image

on:
 release:
types: [published]

jobs:
 push_to_registry:
name: Push Docker image to Docker Hub
runs-on: ubuntu-latest
steps:
 - name: Check out the repo
  uses: actions/checkout@v3

 - name: Log in to Docker Hub
```

```
  uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
  with:
   username: ${{ secrets.DOCKER_USERNAME }}
   password: ${{ secrets.DOCKER_PASSWORD }}

- name: Extract metadata (tags, labels) for Docker
  id: meta
  uses: docker/metadata-action@9ec57ed1fcdbf14dcef7dfbe97b2010124a938b7
  with:
   images: tommonks01/hep-sim

- name: Build and push Docker image
  uses: docker/build-push-action@3b5e8027fcad23fda98b2e3ac259d8d67585f671
  with:
   context: .
   file: ./Dockerfile
   push: true
   tags: ${{ steps.meta.outputs.tags }}
   labels: ${{ steps.meta.outputs.labels }}
```

Listing 3: GitHub action to automate docker build and push to Dockerhub.

We again note that the action will only be initiated on a new release of the code. A release is created through GitHub itself. This is intuitive (achieved by clicking on the new release link on the main page of the GitHub repository), but we recommend users consult up-to-date GitHub documentation first.

Depending on the complexity of the software environment the image build will take a few minutes to several hours. Our experience is that typical simulation environments take minutes rather than hours to build. Once the action has successfully run, a new Docker image will be available to pull from DockerHub. The script above will tag the image twice. It will automatically add the tags *latest* (overwriting the current *latest* version) and <release>. Taking our previous example this would be *tommonks01/hep-deploy:latest* and *tommonks01/hep-deploy:v0.1.1*

4.4 Pulling the updated image to a local machine

Users of the simulation model can now pull the updated model from DockerHub. A pre-requisite is that they must have container software installed on their machine. Docker, for example, will work on Windows, Mac, and Linux. Our DockerHub page has up-to-date instructions on running the Docker container once downloaded. Listing 4 details the commands in order.

```
docker pull tommonks01/hep-sim
docker run -p 8501:8501 tommonks01/hep-sim:latest
```

Listing 4: Commands to run the container

5 DISCUSSION AND CONCLUSION

5.1 Summary and strengths

We have presented an example of a hybrid application using discrete-event simulation with continuous integration and containerisation. By enabling deployment, this is a solution to increase the functionality of free and open source simulation models, and overcomes a significant barrier to sharing models developed

in open source software, which is compatibility across systems and dependency management. Isolating the model and its environment, including software and all required packages, in a container ensures that the simulation model can be used across different computing environments. This means that healthcare users and other researchers can interact with the model without installing software and packages, supporting maintenance, use, re-use.

Our example application used a DES model developed for orthopaedic elective planning, and provided a set of steps to support M&S researchers to use our hybrid deployment framework to convert simulation models from disposable research artifacts to reusable solutions across health and social care. Our example application uses Docker. The model is runnable on Windows, Mac OS or Linux distribution; users simply need to install the correct version of *docker*.

Other methods for sharing simulation models are available, each with advantages and disadvantages (e.g. Harper & Monks, 2023). For example, a simple approach is the use of a package manager (such as pip or conda) to create a *virtual environment*. The package manager is installed directly on the users' machine and will attempt to install the correct operating system-specific version of software packages directly. However, conflicts between packages can still arise, as different packages may have incompatible dependencies or require specific versions of shared libraries, and potentially requires more maintenance and testing. Containers eliminate the need to manually install and configure dependencies on each hosting platform.

For healthcare M&S researchers, a long-standing challenge is translating model results into real-world practice. Using the methods we have described overcomes some of the barriers of model use/reuse in healthcare by enabling decision-makers to access and use the model. As well as providing a consistent environment and portability, containerisation ensures that the application can be reproduced exactly, enabling replication of results, even when deployed on different systems.

5.2 Challenges for the M&S community

Using continuous integration and containerisation naturally presents challenges for M&S researchers. To start, Docker requires the use of FOSS software such as R or Python, as commercial off-the-shelf (COTS) simulation software have proprietary licenses and each container may require a paid software license (that may be time limited in nature such as an annual subscription). COTS packages may also be Microsoft Windows based: an operating system that is typically not used in containerisation solutions due to licensing.

A further challenge is that FOSS deployment comes at the cost of time and effort (Cadwallader & Hrynaszkiewicz, 2022). There is likely to be a learning curve for M&S researchers, as Docker and other tools come with their own set of concepts, commands and workflows. There may be adjustments for FOSS users who are used to using IDEs or notebooks to adopt new tools. Ensuring that everything works seamlessly within a container can require additional effort and troubleshooting. However, the advantages can outweigh these challenges, and more example applications such as ours can help researchers to adjust to the learning curve.

5.3 Limitations

Our example application only demonstrates the use of a DES developed in SimPy and Streamlit, hosted in a GitHub repository, using Docker for containerisation, DockerHub for container hosting, and GitHub actions for continuous integration and deployment. There are many other containerisation approaches and platforms.

One limitation of our hybrid simulation and container approach is persistence of model state and results within the Docker container. When the container is shut down both the simulation model state and results vanish. This means that users must save results or inputs to a file on their own machine and load them again each time the model is opened. If a persistence feature is required a natural extension of our hybrid approach is therefore to include Docker volumes that can persist data long term – these allow Docker to use a local hard drive for storing data.

5.4 Future work

In future work, we aim to develop a framework to test and implement methods to support model deployment for M&S healthcare researchers to increase transparency, longevity, and re-use of models while reducing research waste. This includes helping researchers to overcome common challenges such as time, licensing, technology, and training, and aims to increase the quality and reproducibility of published simulation models.

Code availability

All simulation code, build scripts and tests are available on GitHub: GitHub - Model of orthopaedic ward and archived at Zenodo: https://zenodo.org/record/8011462. The Docker Container can be found here: https://hub.docker.com/r/tommonks01/hep-sim

ACKNOWLEDGMENTS

The authors are supported by the National Institute for Health Research Applied Research Collaboration Southwest Peninsula. The views expressed in this publication are those of the author(s) and not necessarily those of the National Institute for Health Research or the Department of Health and Social Care.

REFERENCES

Allen M, Bhanji A, Willemsen J, Dudfield S, Logan S, Monks T (2020). A simulation modelling toolkit for organising outpatient dialysis services during the COVID-19 pandemic. *PLOS ONE, 15*(8), e0237628-e0237628

Abraham, S., Paul, A.K., Khan, R.I.S. and Butt, A.R., 2020, October. On the use of containers in high performance computing environments. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)* (pp. 284-293). IEEE.

Anagnostou, A., Groen, D., Taylor, S. J., Suleimenova, D., Abubakar, N., Saha, A., ... & Anokye, N. (2022, December). FACS-CHARM: A Hybrid Agent-Based and Discrete-Event Simulation Approach for Covid-19 Management at Regional Level. In *2022 Winter Simulation Conference (WSC)* (pp. 1223-1234). IEEE.

Auer, S. et al., 2021. Science forum: a community-led initiative for training in reproducible research. *eLife* https://doi.org/10.7554/eLife.64719

Byrne, J., Heavey, C., & Byrne, P. J. (2010). A review of Web-based simulation and supporting tools. *Simulation modelling practice and theory*, *18*(3), 253-276.

Cadwallader, L. and Hrynaszkiewicz, I., 2022. A survey of researchers' code sharing and code re-use practices, and assessment of interactive notebook prototypes. *PeerJ, 10*, p.e13933.

Chalk, D., Robbins, S., Kandasamy, R., Rush, K., Aggarwal, A., Sullivan, R., & Chamberlain, C. (2021). Modelling palliative and end-of-life resource requirements during COVID-19: implications for quality care. *BMJ open*, *11*(5), e043795.

Dagkakis, G., & Heavey, C. (2016). A review of open source discrete event simulation software for operations research. *Journal of Simulation*, *10*(3), 193-206.

GitHub (2023) GiHub Actions. Available at: GitHub Actions Documentation - GitHub Docs

Goldberg, A. P., & Karr, J. R. (2020). DE-Sim: an object-oriented, discrete-event simulation tool for data-intensive modeling of complex systems in Python. *Journal of Open Source Software*, *5*(55), 2685.

Harper, A. & Monks, T. 2023. A framework to share healthcare simulations on the web using free and open source tools and Python**.** Proceedings of the Operational Research Society Simulation Workshop 2023 (SW23) C. Currie and L. Rhodes-Leader, eds. https://doi.org/10.36819/SW23.030

Kazil, J., Masad, D., & Crooks, A. (2020). Utilizing python for agent-based modeling: The mesa framework. In *Social, Cultural, and Behavioral Modeling: 13th International Conference, SBP-BRiMS 2020, Washington, DC, USA, October 18–21, 2020, Proceedings 13* (pp. 308-317). Springer International Publishing.

Krafczyk, M., Shi, A., Bhaskar, A., Marinov, D. and Stodden, V., 2019, June. Scientific tests and continuous integration strategies to enhance reproducibility in the scientific software context. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems* (pp. 23-28).

Lauwens, 2021. SimJulia. Welcome to SimJulia! — SimJulia documentation (simjuliajl.readthedocs.io)

Martin-Martinez, E., Samsó, R., Houghton, J., & Solé Ollé, J. (2022). PySD: System Dynamics Modeling in Python. *The Journal of Open Source Software, 2022, vol. 7, num. 78, p. 4329*.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux j, 239(2), 2.

Miller, C., Padmos, R. M., van der Kolk, M., Józsa, T. I., Samuels, N., Xue, Y., ... & Hoekstra, A. G. (2021). In silico trials for treatment of acute ischemic stroke: Design and implementation. *Computers in biology and medicine*, *137*, 104802.

Mohd, S., Mustafee, N., Madan, K., & Ramamohan, V. (2021). Leveraging Healthcare Facility Network Simulations for Capacity Planning and Facility Location in a Pandemic. *Available at SSRN 3794811*.
Monks, T., Currie, C.S., Onggo, B.S., Robinson, S., Kunc, M. and Taylor, S.J., 2019. Strengthening the reporting of empirical simulation studies: Introducing the STRESS guidelines. *Journal of Simulation*, *13*(1), pp.55-67.
Monks, T., Harper, A., Anagnostou, A. & Taylor, S. 2022. "Open Science for Computer Simulation". DOI: 10.31219/osf.io/zpxtm
Monks, T. and Harper, A. 2023. Computer model and code sharing practices in healthcare discrete-event simulation: a systematic scoping review" on the Open Science Framework https://osf.io/c4ytf/
Moreau, D., Wiebels, K. and Boettiger, C., 2023. Containers for computational reproducibility. *Nat Rev Methods Primers* **3**, 50.

Mustafee, N., Harper, A., & Onggo, B. S. (2020, December). Hybrid modelling and simulation (M&S): Driving innovation in the theory and practice of M&S. In *2020 winter simulation conference (wsc)* (pp. 3140-3151). IEEE.
Palmer, G. I., Knight, V. A., Harper, P. R., & Hawa, A. L. (2019). Ciw: An open-source discrete event simulation library. *Journal of Simulation*, *13*(1), 68-82.
Philip, A.M., Prasannavenkatesan, S. and Mustafee, N., 2022. Simulation modelling of hospital outpatient Department: a review of the literature and bibliometric analysis. *Simulation*, p.00375497221139282.
Podman (2023) Available at: Introduction — Podman documentation
Rudyy, O., Garcia-Gasulla, M., Mantovani, F., Santiago, A., Sirvent, R., & Vázquez, M. (2019, May). Containers in hpc: A scalability and portability study in production biological simulations. In 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 567-577). IEEE
Salmon, A., Rachuba, S., Briscoe, S. and Pitt, M., 2018. A structured literature review of simulation modelling applied to Emergency Departments: Current patterns and emerging trends. *Operations research for health care*, *19*, pp.1-13.
Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access*, *5*, 3909-3943.
Shasha, L.U., Haili, X.I.A.O. and Xiaoning, W.A.N.G., 2022. Application of Container Technology in High Performance Computing Environment. *Frontiers of Data and Domputing*, *3*(6), pp.118-126.
Streamlit (2023) Streamlit. Streamlit documentation
Team SimPy (2020) SimPy About SimPy — SimPy 4.0.2.dev1+g2973dbe documentation
Ucar I, Smeets B, Azcorra A (2019). "simmer: Discrete-Event Simulation for R." *Journal of Statistical Software*, **90**(2), 1–30. doi:10.18637/jss.v090.i02.
van der Kolk, M., Miller, C., Padmos, R., Azizi, V., & Hoekstra, A. (2021, June). Des-ist: a simulation framework to streamline event-based in silico trials. In Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part III (pp. 648-654). Cham: Springer International Publishing.
van der Ham, R. (2018). salabim: discrete event simulation and animation in Python. Journal of Open Source Software, 3(27), 767.
Wall, J., Ray, S. and Briggs, T.W., 2022. Delivery of elective care in the future. *Future healthcare journal*, *9*(2), p.144.

AUTHOR BIOGRAPHIES

ALISON HARPER is a Research Fellow at University of Exeter Peninsula Collaboration for Healthcare Operational Research and Data Science (PenCHORD). Her research interests include applied health and social care modeling and simulation, hybrid modeling, and real-time simulation. Her email address is a.l.harper@exeter.ac.uk.

THOMAS MONKS is Associate Professor of Health Data Science at University of Exeter Medical School and Turing Fellow at the Alan Turing Institute. His research interests include open science for computer simulation, urgent and emergency care, and real-time discrete-event simulation. His website is https://arc-swp.nihr.ac.uk/about-penarc/people/tom-monks/ His email address is t.m.w.monks@exeter.ac.uk

SEAN MANZI is a Research Fellow at University of Exeter Peninsula Collaboration for Healthcare Operational Research and Data Science (PenCHORD), focusing on understanding and improving healthcare services, mainly in mental health. His research interests include network-based whole system modelling of mental health services. His email address is: S.S.Manzi@exeter.ac.uk