Plausibly Realistic Sociotechnical Simulation with Aspect Orientation

Tom Wallis

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

School of Computing Science

College of Science and Engineering

University of Glasgow

Table of Contents

Todo list

Chapter 1

Introduction

1.1 A primer on aspect orientation

Explain aspect orientation briefly. Some useful notes for the explanation:

- AO originated in xerox parc, first described in [1]. There are lots of weaving mechanisms for regular, static aspect orientation, and there's a good early survey of them all (and implementation in a custom OO language specifically for this!) in [2].
- AO has some forebears: metaobject protocols, subject-oriented programming, adaptive programming, composition filters. The latter three are described by [3] as being alternative *kinds* of AO — I disagree, but they're certainly attempting similar things.
- The original and still most widely used AO implementation is AspectJ, which comes with its own aspect language. It's grown over the years and is used sometimes in industry [citation needed...]. A smaller alternative would be Spring AOPfind a citation for spring AOP

1.2 Prior Work

Write a section for the introduction describing the work done on pdsf before this, to delineate where we're starting from and avoid any claims of plagiarism. This can be short, the first sec of the lit

review is a proper discussion, but the tool should be mentioned here. See section 2.1 for what already exists.

1.3 Terms & definitions

Complete the glossary in section 1.3.

Decide whether terms like BPMN, simulation & modelling, etc also belong in the glossarysection 1.3.

Aspect

Advice

Joinpoint

Pointcut

Weaving

AspectJ The original aspect orientation framework, with language extensions to describe pointcuts and aspects.

Target The procedure an aspect is applied to via a join point, to affect advice.

PyDySoFu

Chapter 2

Relevant Literature

The work presented in this thesis revolves around the combination of simulation & modelling and aspect orientation. As discussed, some research on the topic was also done prior to this body of work I assume we discuss PDSF existing in the intro. There are therefore several things to discuss: work that was done on this project prior to this thesis must be discussed so as to make clear what is *new* in this thesis, related work by others should make clear the context in which this research was done, and opportunities found in this related work for which PyDySoFu is well-suited must be identified to set the scene for the rest of this thesis.

With this in mind, section 2.1 discusses the earlier work contextualising this thesis, summarised in [4]. As PyDySofu has some unusual features as an aspect orientation library, section 2.2 follows this with a review of similar approaches in the literature. section 2.3 and section 2.4 then review aspect orientation as applied in simulation and modelling respectively, and section 2.5 reviews literature on variable behaviour in simulation and modelling, a key strength of PyDySoFu. Finally, research opportunities identified in these various bodies of work are discussed in section 2.6, which pave the way for the research composing the body of this thesis.

2.1 Early work on PyDySoFu

PyDySoFu[1] is a Python library [5] built for making changes to the source code of a Python function as it is called, and before it is executed, while the original function definition remains oblivious to the changes being made. It was originally developed as an honours-level dissertation, which was built upon and detailled in a subsequent paper [4]. This thesis furthers that original work. To be clear about the work this thesis contains, the state of the project *before* this work began is briefly discussed here.

2.1.1 PyDySoFu's implementation and features

The original version of PyDySoFu[2] patched Python classes with additional handlers. Attributes of Python objects are usually retrieved using dot notation (i.e. `object_id.attr_id`), which evaluates internally to a call to `object_class.__getattribute__(``attr_id'')`. PyDySoFu replaces a class' built-in `__getattribute__()` method with a new one, which calls the original to acquire the required attribute.

In the case where the required attribute is not callable, the value is returned as normal. Callable attributes are modified, however. In this case, the replacement `__getattribute__()` also checks for a set of manipulations to make to the original code. These can be applied before or after the original code is run, as well as around it. A new function is returned containing a reference to the originally sought attribute, but which will search for these additional pieces of work before executing it, and can execute this work before or after the call (or both). These pieces of work are referred to as "advice", adopting aspect orientation terminology.

As discussed further in section 2.1.2, this approach is effectively an implementation of a traditional aspect orientation framework. However, unlike existing frameworks, PyDySoFu also supports a special kind of "around" advice: before a function is called, it can be rewritten. This is done by applying "before" advice which retrieves the abstract syntax tree of the target callable attribute using Python's `inspect` module (its built-in reflection), applying arbitrary transformations to the tree, and recompiling it into a Python `code` object (its representation of its internal bytecode). At this point, many

[1] Or "PDSF" for short.
[2] Further improvements have been made through this research which improve on the design, but this section is to discuss the state of the project before this work began, and the general principles around it, which remain unchanged.

things are possible: the transformation can be cached for later use, can replace the original callable's `code` object to make the transformation persistent, or can be discarded after use. This transformed code is run in lieu of the original, effectively enabling aspect orientation which can make adaptations *inside* a procedure as well as before and after its execution.

This approach also had some limitations:

- Traditional pointcuts cannot target points inside a procedure, meaning that an aspect applied "inside" its target must manage the points where its transformation is applied manually.
- Importantly, a callable object's internal bytecode cannot be replaced in Python3, leading to a rewrite discussed in chapter 3.
- This method is significantly slower than other aspect orientation approaches, as rewriting a class' `__getattribute__` method means that *every* resolution of an object's attributes — whether they are methods or values, and including a class' built-in "magic" methods — incurs an overhead from the replaced `__getattribute__` implementation. However slight this overhead can be made, affecting Python's built-in methods on classes means that rewriting the `__getattribute__` method is unavoidably expensive due to the scale of these methods' use.

However, the goal of the original research was to develop a flexible "proof-of-concept" of aspect orientation adapting procedure definition at runtime, which was successfully achieved[4], [6].

2.1.2 Aspect Orientation & PyDySoFu

The goals of "changing a function's behaviour" and maintaining "obliviousness" in the original definition of that function speak to the goals of the aspect oriented programming paradigm[1]. Quoting their original definitions:

> "Components are properties of a system, for which the implementation can be cleanly encapsulated in a generalized procedure. Aspects are properties for which the implementation cannot be cleanly encapsulated in a generalized procedure. Aspects and cross-cut components cross-cut each other in a system's implementation.

> […] The key difference between AOP and other approaches is that AOP provides component and aspect languages with different abstraction and composition mechanisms."

Generally, aspect orientation is percieved to be a technique for separation of concerns. Any cross-cutting concerns can be separated from their components into aspects applied where that concern arises. The strength of aspect orientation lies in its compositional nature: developers can write short, maintainable implementations of a procedure's core purpose (for example, business logic) and ancillary concerns such as logging or security can be woven into this implementation as preprocessing, compilation, or at runtime. This compositional nature is what gives rise to aspect orientation's "obliviousness", as the procedure targetted by a piece of advice is written without regard to that fact.

The original PyDySoFu implementation was an aspect orientation library focusing on separating a function's definition from *potential changes to it*. This was used to model "contingent behaviour" — behaviour sensitive to some condition — as an original, "idealised" definition of that behaviour, plus some possible alterations. These changes might apply to many different behaviours in the same manner, and therefore represent concerns which separate cleanly into an aspect. An example would be the behaviour of a worker whose job requires focus on allocated tasks. A lack of focus could be represented as steps of the worker's tasks being executed in duplicate, out-of-order, or skipped. Assuming aspects as described by Kiczales, Lamping, Mendhekar, *et al.* are able to edit the definition or execution of a procedure[3], such contingent behaviours are well modelled as aspects.

To achieve this, a model was presented in [4] wherein aspects were developed which could change function *definitions* on each invocation of that function, contingent on program state. This allowed behavioural adaptation to be simulated in an aspect-oriented fashion. In addition, a library of behavioural adaptations called FUZZI-MOSSCITECITECITE was developed which implemented many cross-cutting, contingent behaviours in procedural simulations of socio-technical systems.

One important contribution of this work is that PDSF aspects are effectively able to operate *inside* a target. In typical aspect orientation frameworks such as AspectJ [7], aspects operate by effectively prepending or appending work to a target, referred to as "before" or "after" pointcuts respectively. To do both is referred to as "around". By manipulating procedures within Python directly, PDSF is able to manipulate its target from a new perspective, adding (or removing) work during the target's

[3] As opposed to simply wrapping it with additional behaviour before and/or after execution

execution[4]. Moreover, because weaving is performed dynamically, every execution of a function may perform different operations.

2.1.3 Opportunities presented by PyDySoFu

PyDySoFu presented several oppportunities for future research. Some salient properties of the original work include:

- It provided an aspect orientation library which could weave and unweave aspects during program execution, without relying on anything other than Python's built-in language features. As discussed in section 2.2, this is supported by some early aspect orientation frameworks also, but AspectJ dominates in the world of aspect orientation frameworks and does not support weaving during program execution.
- It provided the capacity to weave aspects *inside* targets, as opposed to around them, or at either end of their execution. So far as we are aware, no aspect orientation framework in research or industry has offered this feature, and its applications and potential are yet to be explored.
- Relatedly, PyDySoFu was used in the context of simulating behaviour which may change over time. Contingent behaviour being a cross-cutting concern is an innovation of the early research which suggests aspect orientation may have strong applications in socio-technical simulation & modelling.

Do we need a brief explainer of what aspect orientation is before jumping into outside lit? Or will this go in the introduction? already a little in the earlier litrev subsections.

The amount of potential investigation which can be done into the dynamic weaving of target-altering / "inside" aspects in simulation & modelling applications is vast. While literature on the complete topic is absent, each individual component of this research angle is well-studied on its own. These opportunities might be related to existing literature through the following questions:

- How does PyDySoFu compare to existing aspect orientation frameworks, particularly those with a focus on dynamic weaving? Related frameworks are summarised and compared in section 2.2.

[4] Similarly to **??**, but in an aspect oriented manner.

- What is the use of aspect orientation in simulation & modelling? How does the approach taken in PyDySoFu 's prior work relate to existing approaches? This will be discussed for simulation in section 2.4, and for modelling in section 2.3.

- Variability is important to capture in any socio-technical model or simulation. How is variability treated in existing literature, and how does this relate to PyDySoFu 's approach? This will be explored in section 2.5.

2.2 Dynamism in AOP

Aspect orientation frameworks have supported "dynamic behaviour" in different ways for a long time. This is largely through a technique referred to as dynamic– or runtime–weaving.

Describe for each framework in this section how it compares to PDSF

2.2.1 Dynamic and static weaving

Dynamic weaving integrates advice into a target program during its execution, as opposed to during compilation or a pre-processing step. The advantage of this is flexibility: dynamic aspect-oriented approaches have been proposed for deploying hotfixes in safety-critical scenarios where software systems cannot be taken offline to apply patches [8], and in adaptive mobile scenarios where software may need to alter its properties in response to its environment [9], or when debugging code to apply potential patches without reloading an entire software system [10].

To meet these needs, software systems need to check for available aspects to weave at any join point, as it is always possible that the set of applied advice has changed since the program last encountered this point. The technique therefore presents a tradeoff compared to traditional (static) aspect weaving, as illustrated in [11]. Chitchyan and Sommerville generalise this tradeoff by describing different mechanisms used to implement aspect orientation into three main categories[5], each with their own strengths:

[5] Drawing from [10], [12] where "PROSE", a particularly influential dynamic aspect orientation library, is detailed.

"Total hook weaving" alters all join points where advice may be applied before runtime, so that during execution each join point "watches" for applied advice. The benefit of this approach is that aspects can be applied at any point at runtime, but this flexibility is bought at the cost of maximum overhead: at all points where weaving *may* be possible, checks for applied advice must be made.

"Actual hook weaving" weaves hooks only to join points that are expected to be in use. This limits overhead from watching for applied advice, at the cost of flexibility: during program execution, advice may be applied or retracted *only at specific points within the system*.

"Collected weaving" weaves aspects directly into code at compilation / preprocessingsurely this isn't dynamic, Ian...?!, so as to collect advice and target codebase into a single unit. This provides exactly the necessary amount of overhead, and in many cases may result in requiring no "watching" for applied advice at all, but this limits a developer's ability to amend advice supplied at runtime.

There is an almost direct tradeoff between the number of potential join points actively checking for applied advice at runtime, and the overhead of dynamism in any aspect oriented framework, with "total hook weaving" providing complete adaptability at the expense of checking at all possible points whether advice is applied.

Another tradeoff could be seen to be the clarity of dynamically woven aspect oriented code. Aspect orientation is already criticised for the lack of clarity as to what woven code will *do* when run, and where weaving can change during program execution, static tools are less useful in making these predictions. Some tools have been produced which do provide tooling for achieving understanding as to what dynamically woven code will do when executed (also called an "Aspect Monitor", as discussed in [10]), but they are often limited or missing from a dynamically weaving framework's implementation (such as [13]). Find more citations for both dynamic weavers with aspect monitors and without. Nanning aspects? Nu?

2.2.2 PROSE

One implementation of dynamic weaving is PROSE[10], [12], a library which achieves dynamic weaving by use of a Just-In-Time compiler for Java. The authors saw aspect orientation as a solution to software's increasing need for adaptivity: mobile devices, for example, could enable a required feature by applying an aspect as a kind of "hotfix", thereby adapting over time to a user's needs. Other uses of dynamic aspect orientation they identify are in the process of software development: as aspects are applied to a compiled, live product, the join points being used can be inspected by a developer to see whether the pointcut used is correct. If not, a developer could use dynamic weaving to remove a mis-applied aspect, rewrite the pointcut, and weave again without recompiling and relaunching their project.

Indeed, the conclusion Popovici, Alonso, and Gross provide in [12] indicates that the performance issues generalised by Chitchyan and Sommerville in [11] may prevent dynamic aspect orientation from being useful in production software, but that it presented opportunities in a prototyping or debugging context.

PROSE explores dynamic weaving as it could apply in a development context, but the authors do not appear to have investigated dynamic weaving as it could apply to simulation contexts, or others where software making use of aspects does not constitute a *product*.

2.2.3 Handi-Wrap

Handi-Wrap[13] is a Java library allowing for dynamic weaving via a third-party language designed for metaprogramming, called MayaDo I want a citation for this? Probably not, but worth revisiting.. At the time of development Handi-Wrap's dynamic aspect weaving feature was novel: the aspect orientation library of note, AspectJ, wove only statically[6], and Handi-Wrap's purpose was to show that DSLs for metaprogramming could pave a way to dynamic weaving.

Baker and Hsieh implemented an aspect orientation framework which is reasonably performant, weaves dynamically, and allows for aspect orientation features to be implemented natively for greater

[6] AspectJ now supports what it calls "load-time weaving" — that is, weaving aspects as classes are loaded into the JVM — but not weaving to things that are *already* loaded, meaning AspectJ still allows for only a particular flavour of dynamic behaviour.

control as compared to Handi-Wrap's then competitor, AspectJ. As a tool, Handi-Wrap demonstrated a promising approach to dynamic weaving, but the project appears to have enjoyed less attention than similar work (such as PROSE, described in **??**).

The technique used to implement Handi-Wrap (implementation via a metaprogramming-specific DSL, Maya) is familiar, in that it shares a perspective on dynamic weaving with early PyDySoFu work. The fuzzers used in [4] applied transformations to abstract syntax trees, not unlike a LISP-style macro. To quote [14] by way of contrast: "`Maya generalizes macro systems by treating grammar productions as generic functions…`".Revisit this inline quote format The two approaches have clear differences. Most notably, PyDySoFu's entire implementation *and use* is performed in Python directly, and Maya's intended purpose is metaprogramming in a more general sense. It is possible that, while Maya provided a useful foundation to explore the dynamic weaving of aspects, its lack of adoption as a language limited handi-wrap's reach; nevertheless, it is encouraging to see another use of metaprogramming for weaving aspects at runtime.

2.2.4 Nu

Some extra things here about Nu, such as [15].

Nu is an aspect orientation framework written in Java which achieves dynamic weaving by way of the Nu virtual machine [15]. This introduced new primitives in Java for the application and removal of aspects: `BIND` and `REMOVE`.Write more NU writeup — requires more citations etc.

2.2.5 Binary Component Adaptation

Binary Component Adaptation[16] (BCA) is a technique for performing adaptations on software components after compilation. Though it works on already-compiled code it does provide dynamic behaviour: the technique can adapt software components via rewriting before or during the loading of its target. Like some aspect orientation techniqueswhich?!, BCA adapts a Java class loader to make its adaptations, but unlike aspect oriented approaches it does not require access to the original source of the software. For scientific simulation purposes, it could therefore be appealing in situations where

adaptations are made to another researcher's simulations — assuming the original source code is not published — or in security settings investigating trust in compilers and runtimes[17]. In the present context of developing socio-technical simulations however, this does not appear to be an advantage, particularly at a time when the source code of software components of research projects are increasingly published.

An important distinction to be made is that BCA provides an example of runtime adaptation, but does not enable an aspect oriented approach and is not developed with separation of concerns in mind. It is presented here as a useful contrast to PyDySoFu: it demonstrates an alternative technique for achieving dynamic runtime source manipulation, even if the lack of separation of concerns means it would not be well applied for this thesis.

2.3 Aspect Orientation in Simulation & Modelling

Having discussed aspect orientation as it is used in a simulation context, it is natural to investigate its use in modelling research, too.

Simulation and modelling are similar topics and are often combined into a single study. However, their goals differ. Simulation typically involves the study of processes or behaviour: there is an expectation that simulations are *executed* or *run*. This often produces data. The intent of modelling is more structural in nature: models are typically observed or analysed to gain insights. Quoting Maria's introduction [18]:

> "Modeling *[sic]* is the process of producing a model; a model is a representation of the construction and working of some system of interest. A model is similar to but simpler than the system it represents.
> […]
> A simulation of a system is the operation of a model of the system. The model can be reconfigured and experimented with; usually, this is impossible, too expensive or impractical to do in the system it represents. The operation of the model can be studied, and hence, properties concerning the behavior of the actual system or its subsystem can be inferred."

Maria's definition implies that to simulate is to operate a model. Whether this model is constructed for the purpose of simulation or for study in its own right, a simplified representation of the system

being studied is implicitly required for any simulation. However, modelling does not imply simulation. Models can be studied for their own merits, and many modelling frameworks exist which are made explicitly for their own study, without regard to their use in simulation[7]. Aspect orientation has seen some study in modelling, particularly for socio-technical modelling, and while aspect-oriented socio-technical modelling is not generally researched with subsequent simulation in mind, an important body of work is still present, and therefore important to discuss.

2.3.1 Aspect Orientation in Business Process Modelling

Aspect orientation for socio-technical systems is particularly well studied in the business process modelling community[22], [23] find more citations for AOBPM

2.3.2 MAML & SWARM

Is MAML/SWARM really modelling, or simulation? Simulation, right?

2.3.3 BPMN & aspect orientation

There's tons here. Go through the remarkable read folder. Particularly anything Claudia Capelli's worked on, see [23], [24]. Also the widely cited work on aspect orientation in BPEL[25], and the work on precedence of aspect application in [26],

2.4 Aspect Orientation & Simulation

The simulation section *badly* needs revisiting.

Surprisingly, little literature exists pertaining specifically to the use of aspect-orientation in a simulation context. Aspect orientation is often applied to modelling as discussed in section 2.3, used to compose a perspective of the world from individual parts, but in a way which isn't necessarily

[7] Consider UML, a well-studied modelling framework which is generally not used for any kind of simulation — depending on the use case, it often cannot be — and for which many alternatives now exist specifically to address this limitation [19]–[21].

executable or able to produce data.

Early in the history of aspect orientation as an emerging paradigm, there was some interest in its use for scientific simulation. [27] discuss that computer simulations require code for both observation of a simulation and the simulation itself, and that misuse of this could cause what is in effect a kind of Hawthorne EffectDoes hawthorne effect need a citation?, where the inclusion of observation code intertwined with simulation code might influence the outcome of an experiment. They suggest that improving simulation technologies could combat this approach. Aspect Orientation, being developed specifically with obliviousness in mind, is an ideal candidate which Gulyás and Kozsik identify.

Much of the literature concerning aspect-oriented programming and simulation focuses on tooling support for aspect-oriented simulation, rather than investigations into its efficacy. For example, attempts have been made to integrate aspect orientation into new tools [28]–[30] or into existing ones [3], [4], [28]. Typically, these papers identify a need for aspect orientation in simulation frameworks — the argument often revolves around a need for increased modularity, and occasionally around better structuring of the simulations themselves[8] — but past a small example to demonstrate how the tooling can be used, little additional development is performed. No significant case studies or refactoring of existing codebases of notable scale are provided. This is important because aspect orientation's main strength is pragmatic in nature. If no real-world testing is conducted, it is hard to conclude that the community's suite of modelling tools contribute anything useful when developing simulations.

Some experiments specifically using aspect orientation in the implementation of process-based simulations also exist[8] include more!. For example, Ionescu, Piater, Scheuermann, *et al.* apply aspect orientation in a nuclear disaster prevention simulation. Their motivation is that code can become complex to maintain over time and changes to the scientific zeitgeist or to regulatory requirements leads to costly technical debt. Aspect orientation therefore allows developers to separate functionality into distinct modules more easily, without disturbing the underlying codebase.

[8] See Chibani, Belattar, and Bourouis's series of papers on the topic [3], [31], [32], which culminate in an implementation of a suite of aspects for simulation purposes. Unfortunately, the work still does not produce significant case studies showing the benefits of the technique in practice. Some empirical measurements are made. A lack of significant real-world evidence that the technique works is a major criticism of aspect orientation as a paradigm [33], however, and the use of these empirical measurements designed for different paradigms as a sign of success hints that satisfactory results in the measurement are being treated as more important than empirical effectiveness, an instance of Goodhart's Law: "when a measure becomes a target, it ceases to be a good measure"[34].refactor this footnote into its own paragraph underneath its currently enclosing one.

2.4.1 Aspect-oriented L-Systems

Aspect-orientation is also applied in other simulation paradigms. Cieslak, Seleznyova, Prusinkiewicz, *et al.* investigated the use of aspect orientation in L-system based simulations [35]. An L-system[36] is defined by a set of symbols, an initial string composed of these symbols, and a set of rules for rewriting substrings. While being a powerful tool for representing fractal structures, they were originally conceived of for plant modelling (and still see the most use in this field).

Cieslak, Seleznyova, Prusinkiewicz, *et al.* note that some details of plant modelling are actually cross-cutting concerns against many plants or families of plants. To represent these, they introduce a new language to describe plant models which makes use of aspect orientation to represent these cross-cutting concerns. They test the approach by representing carbon dynamics, apical dominance and biomechanics as cross-cutting concerns that are integrated into a previously published model of kiwifruit shoot development. Cieslak, Seleznyova, Prusinkiewicz, *et al.* hope that these cross-cutting concerns might work in other models too, but this is untested. The use of an aspect in a new model, when developed for another, seems untested in the community's literature writ large and is a noted omission in the conclusion of this particular work.

2.4.2 AOP and simulation tooling

Here, I should be discussing: japrosim work by chibani [3], [31], [32]; OSIF & OSA in [29], [30];

2.5 Variable Behaviour in Simulation & Modelling

In [4], PyDySoFu was used to model behaviour that changed as the simulation progressed. Behaviour undergoing variance appears in literature from many fields, but some themes stand out. Researchers are often interested in:

- Removing small variations from datasets in order to mine the original process (that real-world actors might be deviating from), referred to as sanitisation,
- Inserting variations so as to produce datasets with

2.5.1 Business Process Modelling & variation in behaviour

Is this going from data to a model? Models to data? Potentially multiple models either way depending on their use? Consider this and possibly restructure.

In real-world business processes, natural variation is difficult to avoid. This is because business processes are inherently socio-technical , and so can be expected to exhibit at least slight variations due to the mistakes of human actors executing those processes. Variations can effectively take two forms:

① Some variations are expected, where predictable shifts in behaviour emerge over time. Examples would be habits forming which deviate from prescribed processes, skipped steps, or paths of a fork in a process becoming effectively ignored as others become the default (essentially producing redundancy in the model).

② Unexpected variations can occur if an actor behaves erratically, information is improperly recorded in a log (and so *seems* to exhibit variance), or if some accident occurs. This appears as random noise in collected data, and is difficult to statically embed in any model, as change might take myriad forms and occur at an arbitrary number of points in the process.

As these two forms of variance must be modelled differently, they are treated differently in a business process model exhibiting variance. Typically the second is treated as noise: undesired and a distraction from a model built to reflect a prescribed process. They are therefore removed via sanitisation, and are discussed in **??**. The first, variation which might be reflective of a model as it can be expected to be *executed* — even when this was not an intended or prescribed version — might be interesting to modellers. These situations might arise, for example, where sociotechnical variance within the context of the broader system is the specific subject of investigation. Degraded modes in these systems are a good example of this [37].

2.5.2 Variations in Process Models

Discussing

2.5.3 Process mining & variation in data

Write a short subsection on the trend of santising data / coping with noisy datasets. Sometimes behavioural variance isn't desirable. Consider making noisy data a subsubsection of **??**.

So, there's some stuff to cite on sanitisation and mining in the presence of noise — see On process model synthesis based on event logs with noise, [38].

Add more stuff to be cited here, at the very least...

Process Mining is a field which necessarily deals with erroneous data. As processes are identified within event logs sourced from real systems, inconsistencies in data collected or execution of a prescribed process results in data fed to a mining algorithm which is at best not indicative of the desired result, and at worst indicative of a different one altogether. As a result, variation in process logs is a subject of active research in the community.

There are two main research efforts involving mining on log data with variance:

① Some researchers look to minimise the impact of log variance on the outcome of process mining. This can be done through the development of mining algorithms which are able to cope somewhat with variance. Many algorithms attempting to solve this problem have been developed, but their effectiveness depends on the kind of variance present and the degree to which those different variances are expressed in the data[38].

② Other researchers look to identify noise in event logs before they are mined, processing them to eliminate any variance before mining begins. This requires classification of noise and the removal of suspect traces[39].

Another perspective on the problem is that noise cannot be successfully eliminated, but that training on empirical noise limits a researcher's control over an experiment. The argument here is usually along the lines that empirical noise is effectively impossible to predict, exert control over, or classify entirely, so any testing of tools using that data is flawed. Without knowing an algorithm's response to specific kinds of variance, a researcher can't compare one approach properly or reproducibly against another. It is therefore important to *produce logs with controlled kinds of variance*, so as to create

a kind of synthetic workspace where algorithms are tested against synthetic data with known kinds and degrees of variance. Once they are reproducibly tested against known good datasets, they can undergo empirical verification by using data captured "in the wild".

Naturally, similar approaches exist outside of process mining, as the requirement for synthetic data is a common one. In potentially sensitive data collected on the public — census or health data for example — there may be a need to publish data which is at least partially synthetic [40]–[42]. Drechsler and Reiter presents an array of simple statistical methods for producing this [41]. Koenecke and Varian note that, depending on the nature of the data *needed* for a given application, different methods are appropriate, meaning a variety of techniques are required [43], and give an overview of methods suitable in economics. MetaSim[44] produces data using probabilistic grammars for training neural nets in a manner naturally resilient to variance by including an appropriate amount via the trained grammar, thereby injecting a guaranteed correct degree of noise. Admittedly neural nets are a common source of synthetic data in modern literature and a research subject with a growing need for training data, perhaps best exemplified in the community's production of another kind of neural net specifically for this purpose: Generative Adversarial Nets, or GANs [45].

Approaches specific to the generation of synthetic event logs are also abundant[20], [46]–[49], and as PyDySoFu's original use was in socio-technical modelling, this is our primary interest. Shugurov paper from 2014 should be checked for citing here too. There's a useful summary in On Business Process Variants Generation (Pourmasoumi 2015).

In [46], [50] Stocker and Accorsi describe a method for injecting variance into synthetic event logs ("traces"). In [46], a method is described whereby security-specific alterations to traces can be made which represent the behaviour of an attacker in some socio-technical system. Variance can be injected by statically manipulating a process before simulating it to generate traces or making modifications to traces after simulation. A supporting tool, "Secsy", is provided in [50].

A similar approach approach is provided in [47], where alterations are made to a process before simulation occurs. In terms of alterations made to the model directly (and not produced traces), Secsy only supports a limited number of operations on a model: transformation of `AND` and `OR` gateways to the alternative kind, and swapping the ordering of modelled activities. The method proposed in

[47] is able to make use of a much broader gamut of alterations, by limiting themselves to mutating only block-structured processes which they represent as "structure trees". Working with a tree-like structure allows for edits to be made which preserve the model's validity, and a table of ten potential — reportedly non-exhaustive — modifications to a model are suggested, far more than suggested in the various works on Secsy.

The authors claim that the limitation of requiring block-structured models does not impact the broad applicability of their approach, as Li claims that around 95% of BPMN models can be represented this way [51]. However, that claim should be held with some scepticism. The citations for this claim are [52] and a paper by Polyvyanyy, García-Bañuelos, and Dumas which is most likely [53][9]. The first work checks 214 process models against a set of patterns which are specifically not formalised, and the second presents some formal work on the translation of process models following the block structure relied on in [47] and [51]. However, the two works never cite each other, the application of the formal translations to the patterns presented is non-trivial, and no further explanation as to the application required appears to be presented in the thesis. One could suppose that the translation of the patterns to block-structuring could be automated by an implementation of the theory presented in [53]. In any case, Li notes in [51] that the requirement of block-structuring on a process model is a limiting factor in the application of their own work in their conclusions, and so the broad applicability claimed in [47] should be taken with healthsome caution. After some exhaustive citation reading we can conclude that neither approach supports effective production of synthetic event logs exhibiting a wide gamut of variances by statically manipulating a BPMN model prior to simulation, although the existence of both methods suggests it would be a valuable research outcome.

In [48], we can find good things I should write about.

In [49], we can find good things I should write about. However after going through it it's clearly more variability in *models*, not variability in *data*. Belongs in another subsection.

① [49] — unread, v interesting

② [46] for secsy, and [50], the associated tooling paper

[9] The citation indicates a paper presented a year earlier than [53], and the author has given talks with the same title and published other works with similar titles — although it is possible a paper with the same name was published a year earlier, and this could change Li's claim, some confidence can be had that this is a simple referencing error or typo.

③ [47] generates synthetic logs with variance, just like secsy, but instead of making edits to the process before simulation using a "structure tree" representation and identifying points suitable for mutation.

④ [48] — unread, v interesting

⑤ [20] — Aalst generating logs from models. No variance but they make the case that synthetic data is needed by the community *and* it's a big name taking a swing, too. Could combine well with [47] to get variance without actually producing new techniques, assuming a limitation of the sim approach to block-structured models (which I think they already impose anyway…)

2.6 Research Opportunities in the Literature

One notable omission from the set of research themes outlined in section 2.5 is that variations on processes aren't well studied in their own right. That is to say, behavioural variation is typically treated as a nuisance to the researcher or practicioner interested in a model or dataset, and the variations and their impact on simulations are not studied on their own. Some work existscite work that focuses on variations specifically, but the majority of this is done in the context of tooling, i.e. the representation of variation for their use in another research context, where they are not the subject.

As an aspect orientation framework capable of runtime adaptation of a target system which can manipulate a join point from inside what are our research opportunities given the above...?

Chapter 3

Rewriting PyDySoFu

Check the most up-to-date pdsf implementation — is it actually on that backup drive?

The work undertaken in this thesis required in improved implementation of old tooling. When previously used, PyDySoFu was a proof of concept which could feasibly produce scientific simulations, but was implemented in a manner which was not optimised for speed (making it a burden for large simulations), lacked granularity in the application of its aspect hooks (hooks could only be applied to entire classes), and most importantly, did not work with Python3 (Python2 support officially ended during this PhD).

This chapter briefly outlines the new implementation of PyDySoFu, discusses improvements made to design and performance, and explains some contributions made to the design of aspect orientation frameworks which addresses some core issues raised with the paradigm. Consider adding references to the sections through this PDSF chapter, depending on how beefy it becomes…

3.1 Requirements for Change

As time wore on with PyDySoFu's original implementation, it became increasingly clear that a rewrite was required. PyDySoFu grew out of an undergraduate project, and accrued technical debt as a result of being written under extreme time constraints with little experience. On revisiting, and

on reflecting on other aspect orientation frameworks (as discussed in section 2.2 and [11]) and the use previously found for PyDySoFu (see [4], [6]), it was clear that there were a series of improvements which could be made in the process of rewriting the tool:

- Before this body of work, PyDySoFu made use of techniques for applying aspect hooks which did not translate to the changes Python 3 made to its object model. In particular, Python 3 changed its underlying object model, using a read-only wrapper class that made the replacement of `__getattribute__` impossible via the previous route.

- PyDySoFu's original implementation made no serious accomodations for efficiency. It could be seen as the "total weaving" described by Chitchyan and Sommerville in [11], and it was not possible to provide additional options to ensure that aspects could be as efficiently woven as possible at runtime given a particular use-case.

- The original PyDySoFu implementation wove onto a *class*, meaning that even properties of the class which were not considered join points were still affected by the weaving, even if in a minor way. Because `__getattribute__` retrieves all attributes including special builtin attributes and non-callable attributes, these are also returned via the modified implementation of `__getattribute__`, incurring an overhead, albeit small, for all attribute resolutions instead of a desired subset.

- The original PyDySoFu implementation made no accommodations for scenarios where fuzzing of source code was applied in a "static" manner. That is to say, where a deterministic modification to source is woven as advice, instead of dynamically modifying source code, the same modification would still be made every time the target attribute was executed, unless caching of results was specifically managed by the aspect applying the change. No optimisations were made pertaining to this, but compilation and abstract syntax tree editing have the potential to be PyDySoFu's most expensive operations.

- Unlike other aspect orientation frameworks such as AspectJ [7], join points could not be specified by pattern. Instead, each individual join point must be supplied as a Python object. This means that, while the target attributes are still oblivious to the advice applied to them, the application of that advice could not be written obliviously.

As a large number of requirements were left unfulfilled by the original implementation of PyDySoFu, a new implementation satisfying them was deemed necessary.

3.2 Python3 Specific Implementation

Replacing `__getattribute__` on the class of a targeted method was no longer viable in Python 3. A replacement method therefore had to be found. For clarity: replacing `__getattribute__` allowed for hooks to be woven (at runtime) into likely future targets for advice. These hooks would then discover and manage the execution of advice around each target. Because advice can be run before and around a target, and dynamic weaving implies that advice could be supplied or removed at any time, we look to intercept the calling of any target, and manage advice immediately before execution. So, the task at hand is to find a method of attaching additional work to the calling of any potential target, before that target is executed. We refer to code woven around a target which manages applied advice as *aspect hooks*.

3.2.1 Abandoned techniques

Rather than "monkey-patching"[1] a new version of `__getattribute__` with hooks for weaving aspects, the rewritten method could be patched to the object itself at a deeper level than used in the original PyDySoFu implementation. This would make use of Python's `ctypes` api to patch the underlying object. Similar work has been done in the python community in a project called ForbiddenFruit [54]. Efforts were made to add the required functionality to ForbiddenFruit — patching `__getattribute__` directly on the object, or "cursing" it in ForbiddenFruit jargon — but this was abandoned as the underlying mechanism is particularly unsafe, Python API changes could render the work unusable in future versions easily, and the implementation would only work with particular implementations of Python (for `ctypes` to exist, the Python implementation must be written in C). Community patches existed for cursing `__getattribute__` which did not work, and attempts

[1] Making on-the-fly changes to object behaviours / definitions by taking advantage of scripting languages' typically flexible object structures, such as objects literally being maps from string attribute / method names to the associated underlying value. Monkey-patching makes use of these simple structures and changes object behaviour by replacing values such as the function object mapped to by the original function's name in the dictionary. This is the method by which PyDySoFu originally replaced `_getattribute__` on a class object.

proved challenging, indicating that this would also be complicated to maintain over time. There are also efficiency concerns with this technique depending on its use: weaving advice around a function would mean monkey-patching the built-in class of functions, which would incur an overhead from running aspect hooks on *every function call*.

Other approaches involved making use of existing Python functionality for interrupting method calls. As PyDySoFu wraps method calls at execution time, what is required is to add functionality to the beginning and end of the execution of a method. Python has built-in functionality for implementing debuggers, profilers, and similar development tools, which provides exactly this functionality, as debuggers must be able to — at any point during execution marked as a breakpoint — pause a running program and inspect call stacks, the values of variables, and so on. As a result, the method `settrace()` allows a developer to specify a hook providing additional functionality to a program. Making use of this also has issues in our case. Most significantly, `settrace()` catches myriad events in the Python interpreter which PyDySoFu may not need to concern itself with, incurring significant overhead. In addition, use of the function overrides previous calls to it, meaning that any debuggers used by a user of PyDySoFu would be replaced with PyDySoFu's functionality, which was deemed untenable. However, it is worth noting that the technique could work in theory, and if future versions of Python allow for multiple trace handlers being managed by `settrace()`, this could provide an interesting approach when implementing future dynamic aspect orientation frameworks.

3.2.2 A viable technique: import hooks

A final available technique was to continue to monkey-patch hooks to discover and weave aspects, via an alternative method which did not make use of `__getattribute__`. This approach would change the use of PyDySoFu slightly to make a comprimise between performance and obliviousness of aspect application: when *importing* a module targeted for aspect weaving, methods which are potential weaving targets are invisibly monkey-patched with a wrapper method with a reference to the original[2] and hooks to detect and run dynamically supplied advice.

An important note for discussing the implementation of PyDySoFu is that almost all Python

[2]Necessary to run the originally targeted method.

functionality operates by use of its "magic methods"[3], which has the affect of making the language an ideal environment to implement dynamic aspect orientation. Our method of adding hooks to modules at import time is an example of this. Python's built in importing functionality is managed by `builtins.__import__`, which receives module names as strings and handles package resolution. By monkey-patching the import system, modules can be modified during the process of importing.

Monkey-patching `builtins.__import__` is as simple as replacing the function object with a new one, which has the effect of changing the behaviour of Python's `import` keyword: because all Python functionality relies on magic methods implicitly, its behaviour can be altered in this way. However, our intent is not necessarily to manipulate *all* modules, but a subset of imports specified by a modeller as suitable for manipulation. If all imported modules were affected, this would include all invocations of `import`, including those made recursively by package implementations, for example. Therefore, it is important to have a mechanism to enable and disable the weaving of aspect hooks on each import (effectively, to enable and disable PyDySoFu's modified import logic).

Decide whether this needs to be more thoroughly broken up / structured…

Include a discussion of *what* gets hooks added using this method…

This can be done through another use of magic methods in a manner which also makes clear to a modeller exactly where aspect hooks are being applied: making use of Python's `with` keyword.

3.2.3 Implementing import hooks

We are interested in manipulating `builtins.__import__` only when imports are made which should have aspect hooks woven. We enable this new import behaviour with a syntax of the form:

```
1 with AspectHooks():
2     import mymodule
```

[3] "Magic methods" are methods beginning and ending with two _ characters. The Python language documentation specifies sets of magic methods and their required function signatures which are used internally to implement functionality — for example, any object with the method `__eq__()` defined can be compared against using the == operator, and the `__eq__()` magic method is run to determine the outcome of the operator. Magic methods support more than operator overloading. For example, anything which defines `__len__()` and `__getitem__()` is treated as an immutable container, and adding `__setitem__()` and `__delitem__()` makes that container mutable. Any class defining `__call__()` is treated as a callable object (not unlike a function). More can be found in the Python documentation[55], although more focused guides exist in the Python community [56].

```
class AspectHooks:
    def __enter__(self, *args, **kwargs):
        self.old_import = __import__
        import builtins
        builtins.__import__ = self.__import__

    def __import__(self, *args, **kwargs):
        # ...replacement import logic for performing weaving...

    def __exit__(self, *args, **kwargs):
        builtins.__import__ = self.old_import
```

Figure 3.1: Magic methods used to enable the `with` keyword usage for PyDySoFu

... which would weave aspect hooks into all functions and (non-builtin) class methods within the `mymodule` module object added to the local namespace of the importing stack[4]. Less formally: importing `with AspectHooks()` applies aspect hooks to all potential targets of advice in the `mymodule` package. The behaviour of Python's `with` keyword is defined by more magic methods: any object with `__enter__()` and `__exit__()` defined can be used here, where `__enter__()` is run at the beginning of the enclosed block, and `__exit__()` when leaving the block.

PyDySoFu caches the original `builtins.__import__` object in an instance of the class, and replaces it with `AspectHooks.__import__`, in its `__enter__()` method. This is reversed by replacing `builtins.__import__()` with the cached object in its `__exit__()` function. The resulting implementation for weaving aspect hooks is satisfyingly uncomplicated, as can be seen in fig. 3.1.

3.2.4 Strengths and weaknesses of import hooks

As a technique for weaving aspect hooks, this new method provides multiple benefits. Application of aspect hooks is straightforward from the perspective of a modeller using PyDySoFu, whose code clearly applies aspect hooks and does so in a legible way for future maintainers, i.e. there is no confusion as to where aspect hooks might be applied. Aspect hooks can be applied to specific modules or every module depending on the use of the supplied `with` statement, allowing for total weaving or actual hook weaving [11] depending on their preferences. Further, performance is optimised at least in comparison to the previous implementation of PyDySoFu, as hooks are weave-able at a more granular level (on the

[4] Python's use of the stack namespace in its importing system means that careless re-importing a module can lead to multiple copies of it in different function stacks, meaning that the same name resolution (such as resolving a class by its name in a module) might, after applying aspect hooks in PyDySoFu, change the behaviour of procedures depending on where they are called. Scenarios where this might arise are deemed unlikely enough that the risk of this design decision becoming troublesome are considered negligable. Still, it would be remiss not to make note of the fact.

level of procedures such as functions or methods, rather than all attributes of a class).

However, there are also caveats of this approach that are necessary to address. As aspect hooks are woven in the new implementation of PyDySoFu via Python's import functionality, any procedure not imported from a module cannot have aspect hooks attached.Consider adding local namespace weaving to pdsf3: should be easy to implement as a cheeky little monkey-patch… However, as aspect orientation is primarily concerned with a separation-of-concerns approach to software architecture, targets are expected to exist in other modules, and we do not consider this to be a significant limitation.

A more significant limitation of the import hook approach is that the object with aspect hooks woven exists in the namespace of the function *importing* the function. In other words, this method makes it impossible for a module to make use of aspect hooks that are woven in an unrelated piece of code. We therefore have a "semi-oblivious" property to our aspect orientation approach: targets of advice are unaware of any adaptations made, but *any code making use of those adaptations must be aware enough to at least apply aspect hooks*[5].

In a manner of speaking, this can be considered to alleviate some concerns with aspect orientation as a paradigm. Aspect Orientation is criticised for making reasoning about programs more difficult [33], [57], [58]. One cause of this is that aspects separate logic from where it is run; Constantinides, Skotiniotis, and Stoerzer 's comparison with the jokingly proposed `come from` statement [58], [59] is a reminder that it can be effectively impossible to understand how a program will execute if the path of execution is not at least linear or clearly decipherable from source code. Aspect orientation as a paradigm inherantly violates this linearity. However, import hooks as implemented in fig. 3.1 present code which can be interpreted in one of only two ways:

① Looking at the original implementation of a procedure, its intended execution is clear. A programmer can make use of this directly and it is guaranteed to behave as expected.

② Any program making use of a procedure imported from a module will see, when the procedure is imported, whether it has had aspect hooks applied. In this case its behaviour is unknown — falling prey to the design flaws discussed in the aspect orientation literature **??????** — but this unpredictability is at least highlighted to the programmer.[6]

[5]Note that once aspect hooks are applied, advice can still be supplied from anywhere in the codebase.
[6]It is worth noting that a third case technically exists, where a procedure is imported from a module which imports that

As a result, while import hooks are somewhat limited in that they are applied specifically to imported code and break the traditional AOP concept of obliviousness in at least a weak manner, these two facts combine to arguably fix a latent issue in the design of the aspect oriented paradigm. The original PyDySoFu implementation was able to modify any procedure in a more traditional, oblivious manner. While this new implementation is clearly more limited as a result, we consider these limitations an overall benefit to the design of the tool, and a contribution to aspect orientation framework design.

3.2.5 Weaving process

Describe the improved process of weaving in PDSF3

Find the newest copy of PDSF3, make a repo for it, cite the new impl

3.3 Discussion

The new implementation of PyDySoFu makes a few contributions, particularly in comparison to the previous version:

- Its new technique of weaving aspect hooks on import, making use of Python's `with` keyword, improves aspect orientation framework design by trading a degree of obliviousness for clarity
- Aspect hooks can be applied with more precision than the previous implementation of PyDySoFu, meaning:
  - Users of the framework can better delineate between total and actual hook weaving
  - Unnecessary overheads from checking dynamically applied aspects at each join point are reduced.
-

Despite this, there is room for improvement in the design of the framework:

procedure from another module. If the latter module contains the implementation and the former applies aspect hooks when it imports, then any program making use of the former module will be importing a procedure with aspect hooks applied implicitly. However, these situations are still visible through simple inspection of these chained imports, where other aspect orientation frameworks might apply an aspect to any join point at any time, without this being obviously discoverable by a programer.

- Caching of applied aspects to join points could be implemented. If between two invocations of a target no changes have been made to the applied aspects, a function object containing the composed aspects from earlier invocations should be run. This would permit runtime aspect weaving with less overhead, as searching for applied aspects need not be performed at every target invocation. Targets should have "changedness" flags which are set every time an aspect is applied or removed from it.

- Our intended use case for aspect orientation for simulation & modelling is in scientific codebases specifically; direct integration with the scientific package ecosystem (which is vibrant in Python's community) should be made. A good initial project would be integration of aspect application in sciunit tests [60].

Chapter 4

RPGLite: A Mobile Game for Collecting Data

RPGLite is a game designed with some special qualities: Kavanagh and Miller have produced PRISM models which can be model-checked to identify ideal play strategies in all game states[61]maybe add more kavanagh citations early on!. Some experiments were conducted around RPGLite to answer the question: "Over time, do players converge on an ideal strategy of play?"Fix formatting and correct wording of William's hypothesis, and mine below.

An alternative question to answer would be, "what strategy of play do players typically adopt", and the related question, "do all players adopt the same strategies?" These are not scientific hypotheses, but interesting questions to ask of a game where "correct" and "incorrect" actions can be categorised. Moreover, Kavanagh and Miller's work can identify the *cost* of an action, allowing for even richer datasets and analyses. It would not be possible to perform actual analyses of player behaviour without real-world player data, however.

To that end, a collaboration was undertaken with Kavanagh and Miller to develop and release a mobile implementation of RPGLite which would collect player data for later analysis. Kavanagh and Miller would get to demonstrate the utility of their model checking in an empirical scenario. We would get to develop models representing player behaviour, and check these models against the collected data. This represents an ideal opportunity to make use of aspect orientation in a new context: a model of naive RPGLite play would be produced which represented random playThe naive version *does* do random play...right?, and aspects could be written which augment the naive model with guesses as to

player behaviour. If the data augmented models generate correlates with empirical data more closely than the naive data, we can dismiss naive play as "realistic", and assume the augmented behaviour. Many aspects can be written representing different styles of play, which might be adopted by different players, a concrete benefit of aspect orientation in modelling & simulation. This chapter discusses the design and implementation of RPGLite for data collection purposes, allowing for discussions of actual experiments — and a more detailled examination of the application of aspect orientation — in chapterscref the chapters on specific experiments at the beginning of the RPGLite chapter..

Add an outline of the RPGLite chapter here.

4.1 An Overview of RPGLite

RPGLite is a simple two-player game played in turns. Each player selects characters independent of the other, with each character having a unique set of abilities and properties, which are generally health, chance of success on attack, and damage dealt on a successful attack. The abilities of some characters necessitate additional properties. Each player selects an "alive" character (one with health greater than 0) to perform their action against a chosen "alive" target (or occasionally targets). A successful attack — randomly determined by chance of successful attack for the selected attacking character — results in that character's unique ability being inflicted on their target[s]. A random player is chosen to take a first move, players may always skip their turn as a valid action, and players continue to take alternating turns until a victor is left with the only "alive" characters.

Eight characters are available for selection, with the following abilities:

Knight Deals damage to an opponent character on a successful hit.

Archer Deals damage to two opponent characters on a successful hit.

Wizard Deals damage to an opponent character on a successful hit, disabling (or *"stunning"*) them for the duration of the opponent's next turn.

Healer Deals damage to an opponent character on a successful hit, and heals themselves or, optionally, the other player character instead (assuming that character is still alive).

Barbarian Deals damage to an opponent character on a successful hit, dealing additional damage if their health is low when attacking.

Rogue Deals damage to an opponent character on a successful hit, dealing additional damage if the target's health is low when attacked.

Monk Deals damage to an opponent character on a successful hit, and immediately takes another turn, until their attack is unsuccessful.

Gunner Deals damage to an opponent regardless of success, dealing additional damage on a successful hit.

Specific details of each character — their health, chance to hit, and damage on hit as well as character-specific details (such as the threshold for additional Barbarian or Rogue damage, for instance) — are defined as a *"configuration"* of RPGLite. Different configurations change the game's *"balance"*, a term referring to the relative strengths of different characters or character pairs. For example, if a configuration leaves many characters with initial health values close to a Barbarian's threshold for additional damage, then they become a very powerful character due to their ability to inflict additional damage. If the Monk's chance to hit is high, the repeated turns it offers can be very advantageous. Character skills can work in concert with each other: choosing a Barbarian and Healer such that the barbarian can be kept at low health for additional damage, but the healer can be used to keep them alive, may be an effective strategy depending on the game's configuration. Kavanagh and Miller found that model-checking a configuration of the game could discover the relative strengths of characters and character pairs when played optimallyCite the correct paper for the game blaancing!.

RPGLite's design has two objectives it must meet. First, that it is interesting to players, which requires that it is approachable and complex enough not to be immediately solvable. This is necessary for real-world data collection, and to demonstrate a design representative of something that could conceivably be a real-world game with an active playerbase. Second, RPGLite's design must be sufficiently simple for model-checking. Model checking is a necessary requirement of design because of our need to identify optimal moves: analysis of player behaviour rests on our understanding of how close to "ideal" players are, and whether players approach ideal strategies over time. This is the crux of the work found in Cite William's PhD thesis for explaining RPGLite design., which relies on a reduced

state space in order to calculate optimal moves, character pairings, and the like.

To demonstrate this state space, note that RPGLite games can have their states described by a set of values: the healths of characters on each team, plus a stunned character. with eight characters having a maximum health value of w, x, y, z[1], two players, and an indicator of which character is stunned — which can be represented as an integer index of the stunned character in the currently-playing team[2]. The entropy of a game is therefore:

$$log_2(w \times x \times y \times z \times 2)$$

Where 2 represents the stunnedness indicator for the current player. The entropy of a typical game is therefore $14 bits$. Simulating gameplay means iterating between these states, which can be time-consuming, but this value should indicate that the game — while embedding design elements which make it enjoyable to play — is also simple enough to thoroughly analyse.

4.2 Implementation of RPGLite

Knowing ideal play is useful, but to understand how real-world players would interact with RPGLite, empirical data needed to be collected. To produce this, we developed a mobile online multiplayer version of the game. Play constituted engagement with an experiment for data collection, and after several months, a database logging player behaviour presented a dataset which could be used, in the case of this PhD, to simulate real-world player behaviour and present a "realistic" environment in which to test the effectiveness of aspect-oriented simulation.

This section briefly describes the details of RPGLite's implementation as a data collection tool. Some lessons learned after reflection on the implementation process were documented for the benefit of others' avoiding our errors[62], might produce a more complete overview of the development.

[1] Maximum health values are dependent on RPGLite's configuration.

[2] Note that stunnedness is valid for exactly one character for one turn, meaning that only one character may e stunned at any time, and the status effect immediately resets, making this integer index of which character is stunned sufficient to represent the stun effect.

4.2.1 Mobile app

As a mobile game, RPGLite's user-facing component was an application, distributed through the Google Play Store on Android and the Apple App Store on iOS. This was developed in Unity, a framework for developing games in C# which can be distributed to almost any platform[3]. Most assets were developed in GIMP, with character designs contributed by a commissioned artist online. Unity allowed for a "WYSIWYG" or what-you-see-is-what-you-get interface builder, with event handlers defined in C# code which would "hook" into events signalled by interface element interactions. User-facing components of the game were largely produced by William Kavanagh, the collaborator on the project and original RPGLite designer. Therefore, in an attempt not to take credit for this work, see Cite William's thesis here for client-side development notes for full details.

Beta testing required user engagement. Apps were deployed to Android and iOS devices of colleagues, who played a series of games to check that game logic was robust enough (and graphic design adequate enough) for final distribution of the game. Beta tests were iterated for 2-3 months, until the game behaved correctly in all edge cases and a final design was settled upon.

Insert client-side photos here, maybe William's photos of the design changing over time...?

4.2.2 API & Server-side Logic

As the data collected ought to be empirical, RPGLite was developed as an online game. This required a server and API for a client to communicate with.

A REST API was developed with Python's *Flask* framework. Endpoints were created for almost all in-game actions, allowing for player search, matchmaking, player profile design, game history and statistics analysis, ranking calculations, login and password reset, implement mutexes on sensitive information, and other in-game activities. The API would also allow moves to be made, and reject erroneous game states or unauthorised input from any malicious input. The API would also send push notifications to an opponent's device when moves were made, which beta testing showed improved engagement significantly.

[3]Meaning that there are technically also versions of RPGLite playable on, say, a games console or web browser.

On each of these actions, data was collected about the action performed, and logged in a database. In addition, in-game activities which required no server-side input but were considered to have potential in later analysis would send data directly to be inputted into our database.

A MongoDB database instance was installed and managed on a University of Glasgow Computing Science virtual machine. The no-SQL nature of the database permitted flexible structuring of the data, and easy analysis of the games' results. The API was also hosted on the same virtual machine. A combination of port access rules and hardening of the database itself prevented unauthorised access to the database, ensuring that the data remained untampered-with.

MORE HERE?!

4.3 Empirical Play and Data Collected

In total, players produced a dataset used in this PhD comprising around 4,000 gamesFind the exact number of games analysed entirely completed[4]. It also includes around 1,000,000 datapoints generated by gameplay or player interaction with the client, such as players checking their history or rolling a dice, although these datapoints are not used in the simulations presented in this simulation. The data is drawn directly from the MongoDB database used to run the game.

Completed games drawn from the MongoDB instance contain many fields, including:

- The history of moves made, and the times those moves were made Confirm moves made includes timestamps, I'm sure it does
- The players involved (by username), and the winning player
- The ELO scores of the players before and after playing the game Confirm that games include both ELO before and after the game is played.
- The characters chosen by each player
- The "score" of each player[5]

[4] "Entirely completed" here means games that ended in a win or lose, not abandoned by players or left unfinished by a player who abandoned the app.

[5] RPGLite's mobile app presented users with a naive scoring mechanism used to rank users on a leaderboard, which some users then used to identify other players of a similar notional skill.

Chapter 5

Simulation Optimisation with Aspect Orientation

With a game deployed to experiment participants and a dataset of empirical play collected, it was possible to determine optimal play in any game state. This entirely separate body of work is documented in another student's PhD thesisCite William's PhD thesis. This dataset leads to further research. If we understand how players *should* play, and we have data to indicate how they *do* play, we can investigate how real-world players might be modelled.

5.1 Aims

Aspect orientation's use in previous simulation and modelling efforts have typically focused on the use of aspects to compose model or simulation detailsThere must be tons of good citations for aspects being used to compose together a simulation / model. Critics of aspect orientation note that the act of process composition makes visually understanding codebases difficult, and so ensuring that a simulation properly models real-world behaviour is made trickier with the introduction of aspect orientation. However, aspect orientation might instead be used to *augment* an existing model, by rethinking what aspects are used to represent.

An alternative use of aspects would be to first build a non-aspect-oriented model of *expected* behaviour, and separately build aspects which describe deviations from this. For example, one might more realistically simulate safety procedures by first producing an idealised, "naive" model of what employees are expected to do, and separately model alterations to prescribed behaviour as an employee's boredom, expectation that checks and balances are unneccesary wastes of their time, and so on — effectively, separating out models of degraded modes[37].

Previous research on the use of aspect orientation to model degraded modes adopted the traditionally claimed benefit of aspect orientation: separation of cross-cutting concerns, allowing for a greater reusability of codebases[4]. A repository of cross-cutting concerns in socio-technical simulation such as boredom was developed as a library to be applied to any future models[63]. However, aspects used in simulation have no intrinsic need to represent concerns that are cross-cutting. Indeed, whether they can be accurately used to represent cross-cutting concerns in simulation is the topic addressed in Add a cross-reference to the chapter on cross-cutting concern simulation accuracy when it exists. Aspects might instead be used to represent *amendments to processes* which deviate from an expected norm, in this case represented by the idealised model aspects are applied to.

To more concretely relate this to the experiment at hand: play of RPGLite can be modelled as players matchmaking, picking characters, and then mutually taking turns until one player's characters are entirely expired. Once a player's characters are dead, new matches can be made. This can continue indefinitely. Lacking a heuristic to select next moves or characters, players might be modelled as picking random moves. However, heuristics for move selection can be added to the naive model of play by way of augmenting the processes already defined through aspects. This approach can be of significant utility in both modelling player behaviour and accurately modelling different players:

① Different players might use their own unique heuristics to model play. Each player's behaviour is therefore well described by separating what play "looks like" to what makes a given player play differently to their peers.

② Different players might lean more heavily on different heuristics, or mixes thereof. Play might be characterised by reliance on experience, on recent games, on knowledge of an opponent, and so on; these different variables can be expected to be weighted differently by each player, adding

complexity to the code which models this individualised play.

③ A modeller might discover a new idea for a heuristic long after developing an original concept for a model. The easiest methods for amending the original model should require the least rewriting of original code. Due to the impact of ②, ideal architectures for an approach such as this should require these heuristics to be defined entirely separately to the base model.

Considering ①, ②, and ③, architectures and paradigms which enable separation of concerns are well-suited to defining alternative approaches to play. Some architectural approaches such as mixins or plugin design patterns might support this structure well, but they typically rely on language features (in the case of mixins) or knowledge of software engineering (in the case of design patterns). Aspect orientation is typically provided to developers as a framework or runtime in a language (such as AspectJ[7] or PROSE[10]) and can require minimal architectural understanding to use: concepts are simple, and the effort of composition is alleviated by the supporting framework or runtime.

The approach makes little use of aspect orientation's significant contribution — cross-cutting concerns — as whether behaviour cross-cuts different parts of a codebase is not of interest in this use case. Instead, aspect orientation is treated as a composition mechanism with a reasonably low degree of technical knowledge required.

5.1.1 PyDySoFu Suitability

Some aspect orientation frameworks do not adequately achieve this requirement. For example, the most influential framework, AspectJ, requires the use of language extensions to define integrate aspect orientation[64], and similar additional complexity is added in seemingly every alternative framework, through the use of bespoke virtual machines, compilers, translators, or languages[12], [14], [65], [66].

PyDySoFu, however, requires very little additional knowledge to use. Its design prioritises simplicity and a shallow learning curve that makes its adoption by researchers without a software engineering background feasible: maybe cut this list of reasons PyDySoFu is fantastic...

→ PyDySoFu is implemented as a pure-python library, meaning that it can be installed through Python's package manager (pip) and imported like any other Python library. No additional

supporting infrastructure is required.

- Aspects in PyDySoFu are simple functions which take as arguments whichever pieces of information are pertinent for the function's use as an aspect[1].
- To weave a PyDySoFu aspect requires only a method call, which returns a `callable` which unweaves that aspect.
- Defining PyDySoFu pointcuts requires only a regular expression matching a method name. This can apply to a wide range of join points if required, but where method names are provided directly, the join point is made clear.
- Additional clarity over where aspects *can* be woven is introduced by PyDySoFu's transparent weaving of aspect hooks, mitigating some of aspect orientation's most prominent criticisms.

PyDySoFu therefore satisfies the requirements of this work well: it offers composition of procedures outside of the scope of an original codebase, makes what is being composed where clear to a programmer, and makes no significant changes to Python as a language (thereby requiring users to specialise in fewer tools).

5.1.2 Proposed Experiment

Aspect orientation's use as a composition tool for model components makes sense in principle, but it is unclear whether the addition of behaviours to a naive model would make the model more "realistic". It is unclear whether the changes made would properly represent what might be empirically observed, and while PyDySoFu's design makes understanding *what* is being composed simpler than other aspect orientation frameworks, a composed model under this paradigm is still split across multiple areas of a codebase, making a visual assessment of whether a model accurately reflects the intended behaviour impractical.

We can confirm whether aspects can realistically represent changes to a naive understanding of the real world by comparing their output against empirical data. For example, if a such a model of behaviour in a system outputs data which correlates poorly against empirically collected data, a change

[1] For example, an "encore" aspect which is woven after a target procedure returns will be provided that target's return value.

to that system would make it more realistic if it improved this correlation, and could be said to be realistic if the generated data appeared sufficiently "close" to the empirical dataset — which here means that the correlation between the two is of statistical significance. Such a change can be aspect-oriented. Therefore, we can see the application of aspects as the application of packages of potential improvements to a base model, which can be verified by way of comparison to known-good datasets.

This is the basis of the experiments in this thesis.

With datasets collected empirically on RPGLite's play, we can build a naive model of play and aspects to apply that should realistically model data from players. This can be used to answer the question:

> *" Can aspect-oriented models be said to exhibit realism?"*

To answer this question, we will produce a model of play, and develop aspects which encapsulate different play styles so as to compare the aspect-augmented datasets and naive datasets against the empirically sourced data. The following subsections detail the naive model developed and aspects applied to this model respectively.

5.2 Naive Model

A naive model of play was developed by separating each stage of the actions taken by players in the client-side app, and separating them into individual procedures. To facilitate the retrieval of most information about a simulation in an applied aspect, the model was written so as to contain the entire simulation state as mutable function arguments. The model was written as a workflow, and state of workflow execution was separated into three components: the actor a function invocation (or "step") repesents activity from; the context of that step in the execution of a workflow; and the context of that workflow's execution in a broader environment. Incidentally, we found this structure to allow a flexible and natural implementation of a procedural simulation, which should translate easily to existing simulation frameworks such as SimPy[67]:

Actor — allows the function to identify the actor performing the activity defined by the function.

This argument is any object uniquely identifying an actor.

Context — allows the function to determine details of the current thread of work being undertaken by the actor. This is necessary because in some simulations, the same actor might pause and resume multiple occurrences of the same activity — for example, they might concurrently play three different matches in RPGLite. As a result, it is necessary to understand the context of the action being performed by the actor in question. This argument can be any object uniquely identifying the context of a piece of work, but should be mutable (such as a class or dictionary-like object) to permit the communication of information across invocations of different action-representing functions.

Environment — an actor's actions are often determined by the global environment they act within. There may be ancillary details to the actor's actions and the context of their particular thread of work which they are undertaken within which are used to determine behaviour, such as a landscape they traverse or other actors they might choose to interact with. Because all actors share access to a global environment, this also provides a message passing space, or a space where actors can set values and flags other actors might look to, should those details be more general than their specific thread of work at a given point in time. [2]

Each simulation step suitable as a join point receives these three arguments at a minimum. Aspects applied to these therefore have access to the entire state of the simulation.

The naive model of RPGLite follows a simple workflow mimicking player interaction with the client-side application used by real-world players. Graphically, it would be represented as a flowchart like so:

TODO FINISH WRITING ABOUT THE NAIVE MODEL'S STEPS

[2] This is different to environments in some other simulation frameworks, such as SimPy[68], where the environment controls scheduling and execution: this structure imposes no constraints such as models of time, and anticipates that any such functionality should be implemented by the programmer. However, an environment such as SimPy's might satisfy a programmer's needs when using this particular pattern.

5.3 Aspects Applied

Players can be expected to select stronger characters as they understand gameplay more. Aspects were therefore developed to track player experience with characters. As players became more familiar with the characters they play, we can hypothesise that they will better understand how to play with those characters, and more often select characters they have success with.

Write about *both* aspects so as to answer the hypothesis : Can aspects be used to generate models of alternative behaviours?

Bibliography

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *European conference on object-oriented programming*, Springer, 1997, pp. 220–242.

[2] H. Masuhara and G. Kiczales, "A modeling framework for aspect-oriented mechanisms," in *Proceeding ECOOP'03*, 2003.

[3] M. Chibani, B. Belattar, and A. Bourouis, "Using aop in discrete event simulation: A case study with japrosim," *International Journal of Applied Mathematics, Computational Science and Systems Engineering*, vol. 1, 2019.

[4] T. Wallis and T. Storer, "Modelling realistic user behaviour in information systems simulations as fuzzing aspects," in *International Conference on Advanced Information Systems Engineering*, Springer, 2018, pp. 254–268.

[5] ——, *Pydysofu*, https://github.com/twsswt/pydysofu, 2018.

[6] ——, "Process fuzzing as an approach to genetic programming," in *Proceedings of the SICSA Workshop on Reasoning, Learning and Explainability*, K. Martin, N. Wiratunga, and L. S. Smith, Eds., ser. CEUR Workshop Proceedings, vol. 2151, Aberdeen, UK: CEUR-WS.org, Jun. 2018. [Online]. Available: http://ceur-ws.org/Vol-2151/Paper_S3.pdf.

[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "An overview of aspectj," in *ECOOP*, 2001.

[8] T. B. Ionescu, A. Piater, W. Scheuermann, E. Laurien, and A. Iosup, "An aspect-oriented approach for disaster prevention simulation workflows on supercomputers, clusters, and grids," in *2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*,

IEEE, 2009. DOI: 10.1109/ds-rt.2009.35. [Online]. Available: https://doi.org/10.1109%2Fds-rt.2009.35.

[9] J. C. Hveding, "An aspect-oriented approach to adaptive systems," M.S. thesis, Institutt for datateknikk og informasjonsvitenskap, 2005.

[10] A. Popovici, T. Gross, and G. Alonso, "Dynamic weaving for aspect-oriented programming," in *Proceedings of the 1st International Conference on Aspect-Oriented Software Development*, ser. AOSD '02, Enschede, The Netherlands: Association for Computing Machinery, 2002, pp. 141–147, ISBN: 158113469X. DOI: 10.1145/508386.508404. [Online]. Available: https://doi.org/10.1145/508386.508404.

[11] R. Chitchyan and I. Sommerville, "Comparing dynamic ao systems," 2004.

[12] A. Popovici, G. Alonso, and T. Gross, "Just-in-time aspects: Efficient dynamic weaving for java," in *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, Association for Computing Machinery, 2003, pp. 100–109.

[13] J. Baker and W. Hsieh, "Runtime aspect weaving through metaprogramming," in *Proceedings of the 1st international conference on Aspect-oriented software development - AOSD '02*, ACM Press, 2002. DOI: 10.1145/508386.508396. [Online]. Available: https://doi.org/10.1145%2F508386.508396.

[14] J. Baker and W. C. Hsieh, "Maya: Multiple-dispatch syntax extension in java," in *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, ser. PLDI '02, Berlin, Germany: Association for Computing Machinery, 2002, pp. 270–281, ISBN: 1581134630. DOI: 10.1145/512529.512562. [Online]. Available: https://doi.org/10.1145/512529.512562.

[15] R. Dyer and H. Rajan, "Supporting dynamic aspect-oriented features," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 2, pp. 1–34, Aug. 2010. DOI: 10.1145/1824760.1824764. [Online]. Available: https://doi.org/10.1145%2F1824760.1824764.

[16] R. Keller and U. Hölzle, "Binary component adaptation," in *European Conference on Object-Oriented Programming*, Springer, 1998, pp. 307–329.

[17] K. Thompson, "Reflections on trusting trust," in *ACM Turing award lectures*, 2007, p. 1983.

[18] A. Maria, "Introduction to modeling and simulation," in *Proceedings of the 29th Conference on Winter Simulation*, ser. WSC '97, Atlanta, Georgia, USA: IEEE Computer Society, 1997, pp. 7–13, ISBN: 078034278X. DOI: 10.1145/268437.268440. [Online]. Available: https://doi.org/10.1145/268437.268440.

[19] D. DORI, "Object-process Analysis: Maintaining the Balance Between System Structure and Behaviour," *Journal of Logic and Computation*, vol. 5, no. 2, pp. 227–249, Apr. 1995, ISSN: 0955-792X. DOI: 10.1093/logcom/5.2.227. eprint: https://academic.oup.com/logcom/article-pdf/5/2/227/6244720/5-2-227.pdf. [Online]. Available: https://doi.org/10.1093/logcom/5.2.227.

[20] A. A. Mitsyuk, I. S. Shugurov, A. A. Kalenkova, and W. M. van der Aalst, "Generating event logs for high-level process models," *Simulation Modelling Practice and Theory*, vol. 74, pp. 1–16, May 2017. DOI: 10.1016/j.simpat.2017.01.003. [Online]. Available: https://doi.org/10.1016%2Fj.simpat.2017.01.003.

[21] P. Wallis and F. Cloughley, *The OBASHI Methodology*. The Stationery Office, 2010.

[22] I. Machado, R. Bonifácio, V. Alves, L. Turnes, and G. Machado, "Managing variability in business processes," in *Proceedings of the 2011 international workshop on Early aspects - EA '11*, ACM Press, 2011. DOI: 10.1145/1960502.1960508. [Online]. Available: https://doi.org/10.1145%2F1960502.1960508.

[23] C. Cappelli, J. C. Leite, T. Batista, and L. Silva, "An aspect-oriented approach to business process modeling," in *Proceedings of the 15th workshop on Early aspects - EA '09*, ACM Press, 2009. DOI: 10.1145/1509825.1509828. [Online]. Available: https://doi.org/10.1145%2F1509825.1509828.

[24] H. S. A. da Silva Jose, C. Cappelli, F. M. Santoro, L. G. Azevedo, *et al.*, "Implementation of aspect-oriented business process models with web services.," *Bus. Inf. Syst. Eng.*, vol. 62, no. 6, pp. 561–584, 2020.

[25] A. Charfi and M. Mezini, "Ao4bpel: An aspect-oriented extension to bpel," *World wide web*, vol. 10, no. 3, pp. 309–344, 2007.

[26] A. Jalali, P. Wohed, and C. Ouyang, "Aspect oriented business process modelling with precedence," in *International Workshop on Business Process Modeling Notation*, Springer, 2012, pp. 23–37.

[27] L. Gulyás and T. Kozsik, "The use of aspect-oriented programming in scientific simulations," in *Proceedings of Sixth Fenno-Ugric Symposium on Software Technology, Estonia*, 1999.

[28] A. Aksu, F. Belet, and B. Zdemir, "Developing aspects for a discrete event simulation system," in *Proceedings of the 3rd Turkish Aspect-Oriented Software Development Workshop*, Bilkent University, 2008, pp. 84–93.

[29] J. Ribault and O. Dalle, "Enabling advanced simulation scenarios with new software engineering techniques.," in *20th European Modeling and Simulation Symposium (EMSS 2008)*, 2008.

[30] J. Ribault, O. Dalle, D. Conan, and S. Leriche, "Osif: A framework to instrument, validate, and analyze simulations," in *SIMUTools2010*, 2010.

[31] M. Chibani, B. Belattar, and A. Bourouis, "Practical benefits of aspect-oriented programming paradigm in discrete event simulation," *Modelling and Simulation in Engineering*, vol. 2014, 2014.

[32] ——, "Toward an aspect-oriented simulation," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 3, no. 1, pp. 1–10, 2013.

[33] F. Steimann, "The paradoxical success of aspect-oriented programming," in *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications - OOPSLA '06*, ACM Press, 2006. DOI: 10.1145/1167473.1167514. [Online]. Available: https://doi.org/10.1145%2F1167473.1167514.

[34] M. Strathern, "'improving ratings': Audit in the british university system," *European review*, vol. 5, no. 3, pp. 305–321, 1997.

[35] M. Cieslak, A. N. Seleznyova, P. Prusinkiewicz, and J. Hanan, "Towards aspect-oriented functional–structural plant modelling," *Annals of Botany*, vol. 108, no. 6, pp. 1025–1041, Jul. 2011. DOI: 10.1093/aob/mcr121. [Online]. Available: https://doi.org/10.1093%2Faob%2Fmcr121.

[36] A. Lindenmayer, "Mathematical models for cellular interactions in development i. filaments with one-sided inputs," *Journal of theoretical biology*, vol. 18, no. 3, pp. 280–299, 1968.

[37] C. W. Johnson and C. Shea, "A comparison of the role of degraded modes of operation in the causes of accidents in rail and air traffic management," in *2007 2nd Institution of Engineering and Technology International Conference on System Safety*, IET, 2007, pp. 89–94.

[38] A. A. Mitsyuk and I. S. Shugurov, "On process model synthesis based on event logs with noise," *Automatic Control and Computer Sciences*, vol. 50, no. 7, pp. 460–470, Dec. 2016. DOI: 10.3103/s0146411616070154. [Online]. Available: https://doi.org/10.3103%2Fs0146411616070154.

[39] H.-J. Cheng and A. Kumar, "Process mining on noisy logs — can log sanitization help to improve performance?" *Decision Support Systems*, vol. 79, pp. 138–149, Nov. 2015. DOI: 10.1016/j.dss.2015.08.003. [Online]. Available: https://doi.org/10.1016%2Fj.dss.2015.08.003.

[40] R. J. Little, "Statistical analysis of masked data," *Journal of Official statistics*, vol. 9, no. 2, p. 407, 1993.

[41] J. Drechsler and J. P. Reiter, "An empirical evaluation of easily implemented, nonparametric methods for generating synthetic datasets," *Computational Statistics & Data Analysis*, vol. 55, no. 12, pp. 3232–3243, Dec. 2011. DOI: 10.1016/j.csda.2011.06.006. [Online]. Available: https://doi.org/10.1016%2Fj.csda.2011.06.006.

[42] D. B. Rubin, "Discussion statistical disclosure limitation," *Journal of official Statistics*, vol. 9, no. 2, p. 461, 1993.

[43] A. Koenecke and H. Varian, *Synthetic data generation for economists*, 2020. arXiv: 2011.01374 [econ.GN].

[44] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler, "Meta-sim: Learning to generate synthetic datasets," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2019. DOI: 10.1109/iccv.2019.00465. [Online]. Available: https://doi.org/10.1109%2Ficcv.2019.00465.

[45] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML].

[46] T. Stocker and R. Accorsi, "Secsy: Security-aware synthesis of process event logs," in *Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, St. Gallen, Switzerland*, Citeseer, 2013.

[47] A. Pourmasoumi, M. Kahani, E. Bagheri, and M. Asadi, "On business process variants generation.," in *CAiSE Forum*, 2015, pp. 179–188.

[48] D. Loreti, F. Chesani, A. Ciampolini, and P. Mello, "Generating synthetic positive and negative business process traces through abduction," *Knowledge and Information Systems*, vol. 62, no. 2, pp. 813–839, Jun. 2019. DOI: 10.1007/s10115-019-01372-z. [Online]. Available: https://doi.org/10.1007%2Fs10115-019-01372-z.

[49] A. Yousfi, R. Saidi, and A. K. Dey, "Variability patterns for business processes in BPMN," *Information Systems and e-Business Management*, vol. 14, no. 3, pp. 443–467, Aug. 2015. DOI: 10.1007/s10257-015-0290-7. [Online]. Available: https://doi.org/10.1007%2Fs10257-015-0290-7.

[50] T. Stocker and R. Accorsi, "Secsy: A security-oriented tool for synthesizing process event logs.," in *BPM (Demos)*, 2014, p. 71.

[51] C. Li, "Mining process model variants: Challenges, techniques, examples," Ph.D. dissertation, University of Twente, The Netherlands, 2010.

[52] L. Thom, M. Reichert, and C. Iochpe, "Activity patterns in process-aware information systems: Basic concepts and empirical evidence," *Int. J. Bus. Process. Integr. Manag.*, vol. 4, pp. 93–110, 2009.

[53] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," in *Business Process Management*, R. Hull, J. Mendling, and S. Tai, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 276–293, ISBN: 978-3-642-15618-2.

[54] C. bibinitperiod github contributors, *Forbiddenfruit*, https://web.archive.org/web/20210515092416/https://github.com/clarete/forbiddenfruit, 2021.

[55] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[56] R.

bibinitperiod github contributors, *Magicmethods (a guide to python's magic methods)*, https://github.com/RafeKettler/magicmethods/tree/65cc4a7bf4e72ba18df1ad17a377d879c9e7fe41, 2011 — 2016.

[57] A. Przybylek, "What is wrong with aop?" In *ICSOFT (2)*, Citeseer, 2010, pp. 125–130.

[58] C. Constantinides, T. Skotiniotis, and M. Stoerzer, "Aop considered harmful," in *In Proceedings of European Interactive Workshop on Aspects in Software (EIWAS)*, 2004.

[59] R. L. Clark, "Linguistic contribution to goto-less programming," *Datamation*, vol. 19, no. 12, pp. 62–63, 1973.

[60] D. H. T. That, G. Fils, Z. Yuan, and T. Malik, "Sciunits: Reusable research objects," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, IEEE, Oct. 2017. DOI: 10.1109/escience.2017.51. [Online]. Available: https://doi.org/10.1109%2Fescience.2017.51.

[61] W. Kavanagh and A. Miller, "Gameplay analysis of multiplayer games with verified action-costs," *The Computer Games Journal*, vol. 10, no. 1-4, pp. 89–110, Dec. 2020. DOI: 10.1007/s40869-020-00121-5. [Online]. Available: https://doi.org/10.1007%2Fs40869-020-00121-5.

[62] W. Wallis, W. Kavanagh, A. Miller, and T. Storer, "Designing a mobile game to generate player data - lessons learned," *CoRR*, vol. abs/2101.07144, 2021. arXiv: 2101.07144. [Online]. Available: https://arxiv.org/abs/2101.07144.

[63] T. Storer and W. Wallis, *Fuzzi-moss*, https://web.archive.org/web/20201018121800/https://github.com/twsswt/fuzzi-moss, 2016.

[64] E. Hilsdale, J. Hugunin, M. Kersten, G. Kiczales, C. Lopes, and J. Palm, "Aspectj: The language and support tools," in *Addendum to the 2000 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '00, Minneapolis, Minnesota, USA: Association for Computing Machinery, 2000, p. 163, ISBN: 1581133073. DOI: 10.1145/367845.368070. [Online]. Available: https://doi.org/10.1145/367845.368070.

[65] H. Rajan, R. Dyer, H. Narayanappa, and Y. Hanna, "Nu: Towards an aspectoriented invocation mechanism," Technical Report 414, Iowa State University, Department of Computer Science, Tech. Rep., 2006.

[66] O. Spinczyk and D. Lohmann, "The design and implementation of aspectc++," *Knowledge-Based Systems*, vol. 20, no. 7, pp. 636–651, 2007.

[67] N. Matloff, "Introduction to discrete-event simulation and the simpy language," *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, vol. 2, no. 2009, pp. 1–33, 2008.

[68] S. developers, *Simpy 4.0.1 documentation*, https://simpy.readthedocs.io/en/4.0.1/, 2021.