Aspectually Enhanced Modelling of Sociotechnical Systems

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 [Kic+97]. 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 [MK03].
- AO has some forebears: metaobject protocols, subject-oriented programming, adaptive programming, composition filters. The latter three are described by [CBB19] 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 AOP find 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 **??** 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

This thesis presents an aspect-oriented approach to simulation and experimental design, tooling to support these endeavours, some empirical assessment of both the paradigm and its application in the domain of simulation and modelling. In particular, the presented research makes use of aspects to model hypotheses and the complexities of variable and unpredictable behaviour in the simulation of socio-technical systems.

Relevant literature in this topic typically comes from a variety of fields which do not overlap significantly, meaning that this review must cover segments of several partly related fields. The presented material is eclectic in nature as a result. We therefore present context for unfamiliar readers as well as a motivating case for the work to follow in three distinct areas, which are, in order :

don't use a description list here, this should be a regular paragraph

Section 2.1, which introduces aspect orientation and gives a background on the field;

Section 2.2, which details some existing approaches to the dynamic weaving of aspects;

Section 2.3, which discusses the use of aspect orientation in simulation and modelling; and

Section 2.4, which outlines literature on the modelling of process variance, particularly in simulation and modelling.

2.1 Aspect Orientation

Often, software engineers have to repetitively handle an issue in some codebase, where the issue is pervasive across many parts of the codebase and is necessarily interwoven through its core functionality. Common examples of this are guarding against unsafe concurrency usage, memory management, and logging. Modularity is considered a key trait of maintainable, flexible and legible programs [Par72]. Modern design techniques often centre around the structure of a program to increase its modularity, with object-oriented programming being the standard approach to designing with modularity in mind in many industrially-relevant languages today. Citation needed?

Approaches to modularity typically section codebases into units of functionality. Concerns such as logging and memory management happen in effectively all areas of a codebase, as a result of engineering needs and the properties of a project's language and environment. As a result, common devices employed with the aim of increasing modularity are unable to strip these "cross-cutting concerns" into some separate, modular unit. Programmers separating these concerns into additional modules are expected to see two key benefits:

① A reduction of *"tangling"*, where program logic essential for a program's intended purpose is intermixed with ancillary code addressing cross-cutting concerns, thereby making essential logic more difficult to maintain;

② A reduction of *"scattering"*, where program logic for cross-cutting concerns is strewn throughout a codebase, making maintenance of this code more difficult.

The existence of cross-cutting concerns therefore expected to make maintenance of both ancillary program logic and a program's core logic more difficult. Addressing this, Kiczales et al. introduced the notion of aspect-oriented programming [Kic+97]. Aspects are best described through their component parts:

→ A *"join point"* defines some point in a program's execution (usually the moment of invocation or return of some function or method).

→ *"Advice"* defines some behaviour, such as emitting a logline, which can conceptually happen anywhere in program execution (i.e. what's defined would typically represent behaviour which cuts across many parts of a codebase).

- An *"aspect"* is constructed by composing this advice with *"point cuts"*: sets of join points that define all moments in program execution where associated advice is intended to be invoked.
- An *"aspect weaver"* then adds the functionality defined by each aspect by adding the functionality defined by its advice at each join point defined by its point cut.

The definition of join points and advice, or how weaving occurs, is a matter left for aspect orientation frameworks and languages to define. In employing the technique, aspect oriented programming aims to separate cross-cutting concerns into aspects, removing the aforementioned repetitive code from the logic implementing a program's functional behaviour so that additional pieces of functionality — logging, authentication, and so on — can be maintained in only one place in a codebase (thereby simplifying their maintenance and comprehension), and remaining program logic can be understood and maintained without the overhead imposed by the previously tangled cross-cutting concerns.

In [Kic+97], Kiczales et al. see these engineering concepts as universal throughout business logic, motivating the aspect-oriented approach for the first time. The authors present an implementation of AOP in Lisp, and compare implementations by way of e.g. SLOC count in an emitted C program to a comparable, non-AOP implementation, with two examples (its use in image processing and document processing). They find the idea — which they note is "young" and note many areas where research might help it to grow — can successfully separate systemic implementation concerns such as memory management in a way that reduces program bloat and simplifies implementation. It is noted that measuring the benefits of their approach quantitatively is challenging.

Tooling followed the theoretical work presented by Kiczales et al. [Kic+97] with a demonstration and subsequent technical description of AspectJ, a Java extension for aspect oriented programming [Hil+00; Kic+01]. AspectJ was introduced to satisfy the research community's need for a tool with which to demonstrate the aspect-oriented paradigm in case studies. The tool is intended to serve as "the basis of an empirical assessment of aspect-oriented programming" [Kic+01]. The library makes use of standard aspect-oriented concepts: Pointcuts, Join Points, and Advice, bundled together in Aspects. They define "dynamic" and "static" cross-cutting, by which they refer to join points at specific points in the execution of a program, and join points describing specific types whose functionality is to be altered in some way. Their paper describes only "dynamic" cross-cutting, but presents tooling support, architectural detail of its implementation, and the representation & definition of pointcuts in AspectJ. AspectJ is compared to other tools for aspect orientation and related decompositional paradigms, and the authors are explicit about their approach being distinct from metaprogramming in, for example, Smalltalk or Clojure.

Filman, Friedman, and Norvig isolate properties of aspect orientation which they assert are definitive of the paradigm [FFN00]. Specifically, they claim that aspect oriented programming should be considered "quantification" and "obliviousness":

"A"OP can be understood as the desire to make quantified statements about the behaviour of programs, and to have these quantifications hold over programs written by oblivious programmers. […] We want to be able to say, "This code realises his concern. Execute it whenever these circumstances hold.

These concepts, alongside "tangling" and "scattering", became core concepts in aspect orientation literature. This is in spite of Filman, Friedman, and Norvig giving no concrete definition of the terms in their original paper (nor citing a source for definition). For the purposes of this thesis, we therefore provide the following definitions of the terms:

Quantification is the property of specifying specific points in a program in which that program should change;

Obliviousness is the property of a codebase that it contains no lexical or conceptual reference to advice which might be applied to it, and of the programmer of a target program that their code may be amended by an aspect programmer.

Filman, Friedman, and Norvig write about aspect orientation *"qua programming language"*, and so theorise around aspect orientation as a paradigm independent of a particular instantiation. They are therefore able to arrive at conclusions about the paradigm in the abstract, and can identify concerns for future investigation for researchers in the field and design goals for developers of aspect orientation tooling. They note:

"B"etter AOP systems are more oblivious. They minimize the degree to which programmers (particularly the programmers of the primary functionality) have to change their behaviour to realise the benefits of AOP. It's a really nice bumper sticker to be able to say, "Just program like you always do, and we'll be able to add the aspects later." (And change your mind downstream about your policies, and we'll painlessly transform your code for that, too.)

Whether obliviously designed aspect-oriented systems achieve their intended goals empirically is outside the scope of their work, and the lack of empirical evidence for this is discussed in section 2.1.2. Claims made such as changing one's mind while developing or maintaining a program and having that "painlessly transformed" — an effect of the aforementioned programmer's obliviousness — is incompatible with earlier writing on modularity. Yourdon and Constantine assert [YC79]:

"T"he more that we must know of module B in order to understand module A, the more closely connected A is to B. The fact that we must know something about another module is a priori evidence of some degree of interconnection even if the form of the interconnection is not known.

Aspect orientation's critics describe similar incompatibilities with existing best-practices [Prz10; CSS04], as well as the lack of empirical evidence for the benefits of obliviousness [Ste06]. Claims about "better" aspect-oriented systems being more oblivious should therefore be regarded as *suggestions* from the literature, and while obliviousness and quantification are useful concepts in discussing research in the field. They also give context for the research community's perspective that obliviousness and quantification are design goals for aspect oriented programmers [**AspectCplusplusDesignImp**; **aspect oriented workflow**; Kel08] (though other researchers suggest they may be best applied in moderation [LC07]).

2.1.1 Alternative methods to Aspect Orientation

Aspect-oriented programming's goal of modularising cross-cutting concerns is shared by other paradigms. Seminal work in aspect orientation makes note of similarities to use of reflection, metaprogramming & program transformation, and subject-oriented programming [Kic+97; Kic+01]. They also observe that other disciplines have introduced "aspectual decomposition" independently.

The example of pre-existing aspectual decomposition by way of diagramming given by Kiczales et al. [Kic+97] is in physical engineering. To give a concrete example from their description, differing types of diagrams when engineering a system such as thermal and electric diagrams of a heater are described as "aspectual" because of the modular nature of the diagrams; though there might be many diagrams of different kinds, they compose together to give an overview of the system being designed.

Similar diagramming techniques have independently arisen in other domains since. The Obashi dataflow modelling methodology[WC10] by Wallis and Cloughley models all possible paths of dataflow through "B&IT" (business and IT) diagrams, where business-specific concerns (people, locations, groups, and business processes such as payroll, stock-check or budgeting) are modelled alongside IT concerns such as applications supporting business processes and the software and hardware infrastructure supporting them. Modelling dataflows in this way allows for a comprehensive understanding of assets and business processes. However, in order to understand how data flows between specific assets within a B&IT, sub-graphs ("DAVs", or Dataflow Analysis Views) denote specific pathways through which data flows between source and sink assets. Alternatively, a B&IT can be viewed as a composition of all possible DAVs within an organisation. Dataflows are therefore broken into different diagramming techniques and specific business concerns can be described independently of others, even if these concerns interact in their dataflow pathways (and, therefore, cutting across each other). Obashi therefore allows for the aspectual decomposition of business processes, through the description of an organisation by individual dataflow analysis views, which compose into an overall model of a system in a B&IT diagram. Obashi models are an instance of aspect orientation which were designed for simplicity and comprehension[WC10; Seo11], but trade this for domain-specificity.

Research conducted by Keller and Hölzle [KH98] investigates solutions to the difficulties involved in the integration of software components and their evolution over time, where those components are re-used with differing requirements. By modifying binaries directly, incompatibilities in a program and one of that program's dependencies can be resolved by way of mutating either after compilation. Their implementation defines a representation for the modification of pre-generated Java class binaries, the output of which can be verified as also being valid Java class binaries. Keller and Hölzle claim that BCA allows for dynamic modification of programs with little overhead. They believe BCA is unique in its combination of features, which include engineering concerns such as typechecking code which is subject to adaptation and its obliviousness to source implementation, as well as guarantees that modifications are valid even for later iterations of the program subject to adaptation.

Other engineering techniques can be used to modularise cross-cutting concerns. For example, metaobject protocols describe the properties of an object's class (including, for example, its position within a class hierarchy) in an adaptable manner [KDB91]. These can be used to implement aspect orientation [Esp04], therefore providing at a minimum the same functionality, though they achieve this through their reflective qualities and are designed with metaprogramming as a primary goal as opposed to modularisation of cross-cutting concerns [KDB91; Sul01]. Multiple-dispatch, where methods on objects are chosen to be run based on the properties of the parameters passed at point of invocation, allows for oblivious decomposition without the need for a weaver [DGM08], although this does not support the goals of aspect orientation in totality. For example, a programmer might want their program to exhibit differing behaviour when methods are called with differently-typed arguments, which is supported by multiple dispatch. However, they might instead want their program to exhibit some additional behaviour whenever a method is invoked, such as logging, but might not want to implement logging alongside the rest of their method implementation for clarity or length reasons. Multiple dispatch therefore offers comparable but different functionality to a software engineer. Engineering patterns such as decorators provide similar functionality to aspects [FBS17], in that cross-cutting concerns can be separated into their own module, but they differ in their approach to obliviousness: decorators annotate areas of a codebase they are applied to, and therefore do not offer obliviousness as aspects do. Add a paragraph after this on subject-oriented programming.

More notes Add notes on subject-oriented programming, bigraphs, maybe holons?

2.1.2 Criticisms of aspect-oriented programming

The growing aspect-oriented programming research community collected both proponents and detractors of the paradigm. The developments in aspect orientation pertinent to the research presented in this thesis are discussed in later sections of this literature review. However, common criticisms of aspect-oriented programming are important to present in two regards:

① Discussions of advancements in the field pertinent to the work presented in this thesis should be understood within the context of some perceived weaknesses in the field, which helps to frame an understanding of literature reviewed in this chapter,

② The presented work addresses some criticisms of aspect-oriented programming, meaning that the criticisms of the paradigm writ large and properties of work published in awareness of those weaknesses will motivate some research presented in later chapters.

An early piece of scepticism in the aspect orientation community is [CSS04], in which Constantinides, Skotiniotis, and Stoerzer see AOP's core concepts as having significant similarities to GO–TO statements, which have historically been the subject of some derision in the literature. [CSS04] is, in spirit, a child of Dijkstra's "Letters to the editor: go to statement considered harmful". The authors note that the notion of unstructured control flow makes reasoning about a program complicated — disorientating a programmer by way of "destroying their coordinate system", leaving them unsure about both a program's flow of execution and the states at different points of that flow — and discuss whether aspect orientated programs can have a consistent "coordinate system" for developers. They note that, while Go To statements are at least visible in disrupted code, the AOP concept of obliviousness makes such reasoning even more difficult than Go To statements, as even the understanding of where and how flow is interrupted is not represented structurally within an aspect-oriented program. They compare aspects to a Come From statement, noting that the concept is a literal April Fools' joke for programming language enthusiasts who claim they've found an improvement over Go To statements. The authors conclude that existing techniques, specifically Dynamic Dispatch in OOP, provide similar benefits without the trade-off in legibility of a program's intended execution.

A similar and more thorough critique of the aspect oriented paradigm can be found in Steimann's "The paradoxical success of aspect-oriented programming" [Ste06]. The concern in this paper is that the popularity of aspect oriented programming — which was nearly 10 years old at time of publication — was founded on a perception that it assisted in engineering more than it was proof that such assistance viable in practice. The author notes that most papers are theoretical in their discussion on tooling, that examples were typically repetitive, and that the community's discussion concerned more what aspect orientation is good for than what it actually is in practice. AOP is compared against OOP, AOP's claimed properties and principles are examined in detail, and the impact on software engineering is reasoned about from a sceptical perspective, comparing claims such as improved modularity against classic papers on the subjects (such as Parnas' work on the same). The paper presents a philosophical examination of aspect orientation, assessing the paradigm against its purported merits and discussing whether we should expect, rationally, that the claims made by the AOP research community would hold true. The paper ends noting some benefits of AOP that do hold true under rational scrutiny, and notes that the true utility of AOP may be very different to those purported by the community. Overall, the paper is a philosophical and critical reflection on the state of AOP research and the community's zeitgeist at the time, claims around which are not well-evidenced in literature. In particular, the author sees AOP's promise of unprecedented modularisation as unfulfillable.

Similar sentiments are shared by Przybylek [Prz10], who looks to examine aspect oriented programming within the context of language designers' quest to achieve maintainable modularity in system design. They frame the design goals of aspect orientation as being to represent issues that "cannot be represented as first-class entities in the adopted language". The paper discusses whether the modularity offered by aspect orientation actually makes code more modular. In particular, they distinguish between lexical separation of concerns and the separation of concerns originally discussed by Dijkstra and Dijkstra in "On the role of scientific thought". They assess the principles of modularity — modular reasoning, interface design, and a decrease in coupling, for example — and find that from a theoretical perspective, there are many reasons to believe that the aspect-oriented paradigm can detrimentally impact the expected benefits of proper modularisation in a program. They conclude that the benefits touted by AOP are a myth repeated often enough to be believed, but point to many papers which suggest improvements to the standard AOP approach which might reduce it's negative impact or make it more practically viable. Przybylek presents a critical review of aspect orientation literature, but often hints at others' solutions to the problems identified too.

2.2 Dynamic Aspect Weaving

One implementation of dynamic weaving is PROSE [PGA02; PAG03], 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 misapplied aspect, rewrite the pointcut, and weave again without recompiling and relaunching their project.

Indeed, the conclusion Popovici, Alonso, and Gross provide in [PAG03] indicates that some performance issues 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.

The performance issues noted by Popovici, Alonso, and Gross are explored in more detail by Chitchyan and Sommerville in [CS04]. Chitchyan and Sommerville present a review of early dynamic aspect orientation techniques. The paper reviews AspectWerkz, JBoss, Prose, and Nanning Aspects through the lens of the authors' prior work on dynamic reconfiguration of software systems and their generalisation of dynamic aspect orientation approaches:

① "Total hook" weaving, where aspect hooks are woven at all possible points;

② "Actual hook" weaving, where aspect hooks are woven where required;

③ "Collective" weaving, where aspects are woven directly into the executed code, "collecting the aspects and base in one unit".

Because of the paper's focus on software reconfiguration (rather than the mechanics and design of dynamic aspect weaving specifically), the analysis of the tools presented in the paper is of less relevance to the work presented in this thesis than their generalisation of dynamic weaving. The trade-offs of the three generalised philosophies are discussed. Chitchyan and Sommerville propose that total hook weaving allows flexibility in the evolution of a software product, at the expense of the performance of that product; this contrasts collected weaving, which shifts overhead out of the codebase and into the maintenance effort. Actual hook weaving is positioned as a compromise between the two, offering the best approach for none of their criteria but never compromising so much as to offer the worst, either.

This suggests merit in a tool designed to flexibly offer any weaving approach appropriate for the task at hand. It's explicitly noted that, in practice, one could use many of the systems they describe. Though the paper is an early publication in the field, no tool the authors review offers all three, and none offers collective weaving alongside either kind of hook weaving.

In contrast, Gilani and Spinczyk note that, while there are different approaches to dynamically weaving aspects, no approach is suitable for an embedded environment. This is due to these systems' low power and available memory. Gilani and Spinczyk therefore propose a framework through which weavers can be assessed for suitability in a given environment, or generated from a set of possible features (where, presumably, features would be enabled and disabled as per an environment's needs). Their families of weavers are defined by the similarities of the requirements in domains they are applied to, and specifically defined by their trade-off between dynamism and resource use (asserted to be broadly proportional). It is unclear that a carefully crafted "actual hook weaver", or JIT-compiled "collective weaving", in the parlance of Chitchyan and Sommerville (see [CS04]), would be meaningfully less efficient than static weaving in all but the extreme application areas outlined in the paper (embedded systems with resources in the range of 30kb memory).

Aspect oriented programming's criticism can often be that it doesn't know what it "aims to be good for", and its application in such extreme environments is arguably mistaken from the off. The families outlined in Gilani and Spinczyk's publication are unnecessary if dynamic aspects are not required in their target environments. Steimann's critique of aspect-oriented programming, contrasted against these families, presents an interesting question. If the goals of dynamism and resource efficiency are at odds, and Steimann's stance that aspect-oriented programs do not earn its proponents' plaudits in practice don't be fancy here , what can dynamic aspect weaving be appropriately applied to? In what environment does the trade-off presented by dynamic weaving not necessitate a theory like Gilani and Spinczyk's in the first place? Arguably, that environment is not found in low-resource systems, and a take-away of [GS04] could be that researchers should seek other contexts in which to apply aspect oriented programming. [1] Note from discussion with Tim to work into this: "the anticipated benefits of aspect-oriented programs are not observed in practice" Both paragraphs on Gilani2004 need to be heavily edited.

[1]As discussed alongside [GK99] in section 2.3, simulation & modelling might present a more appropriate field.

et al. propose a new aspect-oriented invocation mechanism, which they call "Bind" [+06]. Bind's design is motivated by perceived opportunities to improve modularity from a design perspective. The impact of "scattering" and "tangling" in a codebase after weaving in some aspect orientation implementations leads to a more complicated post-weave codebase, which in turn leads to increased difficulty including the compilation of aspect-oriented code and the development and execution of unit tests on said code. In order to demonstrate Bind's approach to simplifying post-weave codebases, the design of "Nu", an aspect orientation framework in .NET supporting Bind, is explained and an implementation presented. They present Bind as an alternative to the weaving of aspect hooks (for load-time registration) into target code, in the style of PROSE (see [PGA02; PAG03]), or the weaving of calls directly, in the style of AspectJ (see [Kic+01]). Bind's model for aspect orientation is to apply or remove implementations of cross-cutting concerns to arbitrary sets of join points at a time of a developer's choosing. Nu's model for this is designed with the aim of granularity of join point specification. What results is a flexible model for aspect orientation which is demonstrated to satisfactorily emulate many other models for aspect oriented programming, such as the models of AspectJ, HyperJ, and Adaptive Programming. It is noted that it is "very common in aspect-oriented programming research literature to provide language extensions to support new properties of aspect-like constructs", and note that their work is similar to (yet distinct from) weaving approaches in run-time & load-time weaving, support for aspect orientation directly in a language's virtual machine, and work towards general models of aspect oriented programming (models which can represent a variety of existing approaches). Their approach is flexible and considered enough to warrant impact in the introduction of aspect orientation within virtual machines (because they require no direct support), and in their ability to represent different weaving approaches, arguably *because* their approach is general enough in design to approach the general model worked towards, which qualifies their satisfaction of their motivation to provide a model distinct to the approaches initially discussed. In line with the complaints of AOP's critics, this does not seem to qualify the satisfaction with which they achieve their practical engineering goals.

Relatedly, Dyer and Rajan explain in more depth than in the design and implementation of the Bind mechanism and the implementation of the Nu framework [DR10]. A more technical discussion is presented, in particular on implementation details including optimisation and benchmarking, largely against AspectJ. Notably, the implementation discussed is a Java implementation, rather than the

.Net implementation presented in [**rajan2006nu**]. Many aspect orientation frameworks are language-specific; the existence of Nu's implementation on multiple platforms highlights the work's most interesting facet being the design of the Bind primitive, rather than the framework itself. In a research area where tooling papers are common but the lack of design philosophy & analysis of case studies is frequent fodder for critics, the novelty of the Bind mechanism distinguishes this series of papers.

2.3 Aspect Orientation in Simulation & Modelling

Aspect orientation as applied to simulating systems, and building models of systems, has been researched from several approaches.

A very early example of aspect orientation in simulation & modelling is [GK99]. Gulyás and Kozsik observes that, in the study of complex systems through software models, the software developed typically serves two purposes: the experimental subject, and the observational apparatus used to conduct the experiment itself. Arguing that the separation of these roles ought to make both the implementation of an experimental system and its later analysis simpler, Gulyás and Kozsik proposes the use of aspect orientation as a means of separating what they perceive as cross-cutting concerns of systems modelling. They present their Multi-Agent Modelling Language, a language implemented in Objective-C via the Swarm simulation package and designed for aspect-oriented simulation of agent-based models. Their aspect orientation effectively makes use of Observer patterns to measure a pre-constructed system under simulation, without the observations being an intrinsic component of the simulated system. They find that AOP provides an intuitive and straightforward method by which simulated experimental systems can be composed, and that MAML's simplicity and its philosophy on modelling are more "satisfactory" than Swarm's standard approach, though the paper betrays that its implementation was more complex than initially conceived: the `patch` unix tool was intended for use as their weaver, though the team eventually developed a transpiler from MAML to Swarm instead (which they name `xmc`.). The deciding factors for the development of a custom transpiler are not discussed.

Gulyás and Kozsik's work presents not only tooling for aspect oriented simulation, but some reasoning & philosophy on the potential benefit of using aspect orientation in these endeavours that extends further than the conclusions of modularity through separation of concerns and a reduction of tangling & scattering. In particular, their work discusses specific scenarios in which the *type* of separation of concerns offered by aspect orientation is desirable, and the engineering approach to achieving the aim reasonable. This distinguishes the work in comparison to many aspect orientation papers reviewed in this chapter. Many papers describe the expected benefits by simply drawing from existing literature such as [Kic+97]. The fact that a rare example of detailed reasoning about the appropriateness of aspect oriented programming in a particular domain is highlighted because the domain in question is simulation & modelling; the subject of this thesis. That aspect orientation might be well suited to separating observer and experiment motivates, in part, this thesis' work showing the plausible realism of a simulation in which behaviour is modified by aspects. Put another way: this thesis draws on the idea that, in response to Steimann asking what aspect orientation is good for, Gulyás and Kozsik would seem to answer, "simulation & modelling", a premise this thesis shares.

Chibani, Belattar, and Bourouis discuss two issues in object-oriented programming: "tangling", where separate design elements of a program are woven within each other in program source, and "scattering", where a single design element is strewn throughout the source, rather than being contained within a single area of the codebase. Remove the description of tangling and scattering from here and include in a review of more foundational AOP literature early in the lit review. They don't belong buried so far down in this chapter. They propose that aspect orientation solves these problems, and identify that there are potential benefits in discrete event simulation code in both regards, making DES frameworks with aspect oriented primitives a potentially fruitful contribution to the research community. Chibani, Belattar, and Bourouis identify cross-cutting concerns in DES codebases, including event handling, resource sharing, and the restoration of a simulation run. The contribution of the paper is the discussion of AOP's potential application to DES codebases, and detail of the avenues available for research in the field. Japrosim is presented as a motivating example of an existing DES framework which they see as ripe for the aspect oriented enhancements they identify.

In a later publication [CBB19], Chibani, Belattar, and Bourouis identify opportunities for the use of aspect orientation in simulation tooling, aiming to increase "modularity, understandability, maintainability, reusability, and testability" by applying the paradigm [CBB19]. They present a case study of an application of aspect orientation to simulation tooling by identifying cross-cutting concerns in Japrosim, a discrete event simulation framework, and propose an aspect-oriented redesign of the tool using AspectJ. Chibani, Belattar, and Bourouis describe Japrosim's existing object-oriented design, followed by aspect oriented variations of some design elements, including concurrent process management and in Japrosim's graphical animation features. A similar survey of areas in which Japrosim's source might benefit from the application of aspect orientation is presented by Chibani, Belattar, and Bourouis in an earlier work [CBB14]. In both cases, the main contribution noted is the design itself. Counting the main improvements between the presented aspect-oriented design and the existing object-oriented one is left to future work in the authors' later publication [CBB19], although a concrete implementation is linked to and some quantitative evaluation of that implementation presented in their earlier publication [CBB14]. The quantitative evaluation provides measurements based on Martin's object-oriented design metrics and demonstrates a greater independence of packages in their aspect oriented version of Japrosim than in the original. However, the intended aim of aspect orientation is not to decouple existing packages, but to isolate those packages' cross-cutting concerns into new ones. It is therefore unclear that their quantitative evaluation achieves its improvements as a result of aspect orientation. No further discussion of their results is provided, and it is possible that the improvement is due to the rewriting necessary in their maintenance of the Japrosim source, rather than due to their use of aspect orientation specifically.

In a manner similar to Chibani, Belattar, and Bourouis's [CBB19; CBB13; CBB14], Aksu, Belet, and Zdemir observe that there are opportunities to be found in a simulation framework able to take advantage of aspect orientation [ABZ08]. Examining the DEVS framework Simkit, their proposal for aspect-oriented programming adoption is two-fold: refactoring of the framework itself and aspect-oriented tooling for use by modellers, who represent cross-cutting concerns within their models. Opportunities for improvements in production and development are discussed, and some implementation notes are detailed, although no concrete implementation or evaluation is provided; the work instead proposes design alterations, and the authors "leave it as a future work *[sic]* to explore the usability and efficiency" of aspect orientation used idiomatically alongside Java's existing reflection offerings. The existence of multiple attempts to refactor differing simulation packages with aspect orientation indicates potential for modellers in the use of aspect-oriented patterns, but the real-world utility of the techniques are omitted. Chibani, Belattar, and Bourouis and Aksu, Belet, and Zdemir both seem to defer to the zeitgeist wisdom of the aspect orientation community in their unproven claim that it improves modularity and maintainability of a codebase. [2]

Neither Gulyás and Kozsik nor Aksu, Belet, and Zdemir detail a case study of their techniques with real-world examples. However, Ionescu et al. do in their work identifying an increased demand for computational power in simulation execution on supercomputers [Ion+09]. Existing known-good models might be unsuitable for the extreme requirements of code efficiency modellers contend with, but running the code on different environments requires modifications for suitability in different environments, around which there are regulations and risks of a reduction in quality during maintenance. The authors propose an aspect-oriented solution to the problem, where aspects modify the simulation codebase with minimal overhead. An implementation of a real-world model for disaster prevention is presented, and assessed both by comparison against an equivalent non-aspect-oriented codebase and by assessment of the aspect-oriented variant's scalability and reliability in both cluster and multi-cluster environments. They find that a comparative analysis of generated code and of their simulations in various configurations both indicate that their simulation's aspect-oriented implementation is suitable for use in disaster prevention, implying that aspect orientation could be suitable in scenarios with comparable requirements.

That the authors can conclude that aspect orientation is suitable in a real-world use case constrained by the requirements of supercomputer use seems promising for the aspect-oriented paradigm, which is criticised for its lack of practical evaluation [CSS04; Ste06; Prz10]. As it is therefore a rare example of aspect-oriented case studies, their evaluation methodology is important to highlight. Their code analysis makes use of significant lines of code as a core metric, which doesn't reliably reflect code quality; quoting Rosenberg's "Some misconceptions about lines of code" [Ros97]:

> "(...) the best use of SLOC is not as a predictor of quality itself (for such a prediction would simply reduce to a claim about size, not quality), but rather as a covariate adjusting for size in using another metric."

This is important because a common claim in the aspect-oriented literature for which there is little empirical evidence is that aspect orientation improves the "quality" of a codebase, but the related claims made by Ionescu et al. [Ion+09] are unreliable.

[2] Chibani, Belattar, and Bourouis do present some quantitative evaluation, but this is flawed as previously described.

It is important to note that improvements in code quality specifically are those which have come under scrutiny by the critical papers reviewed in section 2.1.2. The results presented by Ionescu et al. do not satisfactorily address the requests for empirical evidence of improved code quality in these reviewed criticisms. This does not impact their aspect-oriented models' suitability given the motivations of Ionescu et al., which are that models should be amendable for new supercomputing settings without lack of performance. The models described in this work satisfy that aim. These models are also evaluated by way of performance, an important factor in supercomputer use where execution time is financially expensive and power-intensive. Quantitative evaluation of their simulation's execution time shows less than 5% slowdown compared to a non-aspect-oriented implementation. Ionescu et al. deem this a reasonable trade-off for the engineering improvements they observe. Their application to the supercomputing & disaster prevention simulation domains therefore seem satisfactory by way of performance, and meet Ionescu et al.'s aim of demonstrating a modelling technique which permits adapting existing models for use in new environments without directly maintaining the original model's source.

This result is notable with regards the findings presented in this thesis, which similarly aim to alter a pre-existing model without directly altering it, although for purposes such as model reuse and simplification of experimental design rather than for avoiding the regulatory overhead and financial cost of maintaining the models described by Ionescu et al. [Ion+09].

2.3.1 Aspect Orientation & Business Process Modelling

Several projects within the business process modelling research community make use of aspect orientation to design modelling languages which produce less monolithic business process models [Cap+09; Sil+20] and simplify the composition of models [CM07]. As business processes are inherently sociotechnical and this thesis presents tooling for and results in the modelling and simulation of socio-technical systems using aspect-oriented techniques, it is important to review this community's literature. Editing tip from Tim: don't call things "important", this implies other things aren't. Rework areas where I do this. This field is particularly relevant as work on this project prior to this thesis' research models software engineering processes that are conceptually similar to business process modelling (see chapter 3), and there also exists interest in modelling behavioural variance within the business process modelling community (see section 2.4), which is highly relevant to this thesis' concern with the representation of alteration to process and modelled behaviour as aspects.

Charfi and Mezini see opportunities in integrating BPEL, an executable business-process modelling language, with aspect-oriented concepts [CM07]. This is because when BPEL systems are composed together the static nature of the logic being composed is not always appropriate for BPEL's use cases. The specific use-case examined is web service definitions, where changes affecting composition of multiple component parts can affect many areas of a final result, making modification error-prone. They specifically seek to support dynamic workflow definitions — "adaptive workflows" — which BPEL's existing extension mechanisms do not sufficiently support, but the aspect-oriented literature discusses at length (an overview of which is presented in section 2.2). Therefore, they look to construct an aspect-oriented BPEL extension. Using the case study of modelling a travel agency's web services, they create an aspect-oriented extension by first defining how such an extension would be represented graphically in BPEL's workflow diagrams. Further detail is added to arrive at a technical definition with XML representations, weaving mechanics, and eventually the construction of a BPEL dialect, AO4BPEL. The authors find that their pointcut system (which describes join points on both processes and BPEL messages), support for adaptive workflows, and aspect-oriented approach to workflow process modelling make AO4BPEL unique at the time of publication, though related AOP implementations exist in each individual area of their contributions. The work is weakened by brittle semantics around pointcuts, join points, and the temporal nature of workflow modelling. For example, they note that defining contingent behaviour — only applying an aspect conditionally, based on a trace through a simulation of a modelled system — would allow the application of advice only when model state deems this appropriate.[3] They also call for more generally theoretical AOP research, which mirrors the requests some critics of aspect orientation research make (as noted in section 2.1.2).

In a PhD thesis describing AO4BPEL in detail [CM06] Charfi and Mezini presented a generalisation of the notation developed for AO4BPEL, which applies to any graphical workflow modelling language. Accompanying this are some examples of its use building a framework for enforcing certain requirements of BPEL models, and use of that framework to develop aspect-oriented frameworks for enforcing security and reliability within AO4BPEL models.

[3] The contingent application of model adaptation is a motivating case for some work presented in this thesis; see chapter 3 for a discussion.

In later work, Charfi, Müller, and Mezini define a similar aspect-oriented dialect of BPMN they name AO4BPMN [CMM10], after asserting that the concerns addressed by AO4BPEL [CM06; CM07] in the field of executable process languages also apply to business-process modelling languages, and can be solved similarly. The generalised notation of aspectual workflow models presented in Charfi and Mezini's thesis [CM06] are applied to BPMN to produce an aspect-oriented language specifically for process modelling, as opposed to executable business process modelling.

Cappelli et al. also note that cross-cutting concerns exist in business process models, and are specifically motivated by monolithic design approaches common in business process modelling languages. Like Kiczales et al., they claim that a lack of modularity in business process models leads to cross-cutting concerns scattered throughout a model [Cap+09]. To alleviate the issue, they propose a meta-language, AOPML, which incorporates aspect orientation in a metamodel of business process modelling languages, and instantiate it within their own dialect of BPMN. Using a model of a steering committee as a case study, and separating cross-cutting concerns such as logging, the paper proposes reducing complexity and repetition graphically, thereby in a manner more in keeping with the language design philosophies of popular business process modelling languages, the design and use of which are typically graphical [Obj11; DOR95; Obj15]. They note that this is in contrast to other applications of aspect orientation in business process modelling — specifically AO4BPMN — where aspect definitions are written in XML concern not only the advice to be applied but also their relevant join points, as in general programming aspect orientation implementations such as AspectJ. In this way, the AOPML exhibits the spirit of business process modelling more stringently than does Charfi and Mezini's notation for aspect-oriented workflow modelling.

The difference between Charfi, Müller, and Mezini's approach in designing AO4BPMN [CMM10] and Cappelli et al.'s approach in designing AOPML [Cap+09] highlights design decisions taken when introducing aspect orientation in a new domain. There is an opportunity for a domain-specific aspect orientation framework to align its design with the traditions and idioms already present in models within that domain, but doing so may break the traditions and idioms which already exist in aspect-oriented approaches in other domains. Comparing the approaches of Charfi, Müller, and Mezini and Cappelli et al. does surface that there may be no clear "best" design approach when blending pre-existing modelling paradigms, such as the graphical modelling languages used in business-process modelling and the abstract concepts of aspect orientation. The discussion around whether it is more desirable to adapt existing design elements of aspect-oriented frameworks to a given domain or adapt that domain's existing modelling traditions and idioms to incorporate aspect orientation as it is used elsewhere is outside the scope of this thesis.

New concepts within the design of aspect orientation frameworks are addressed in the business process modelling community. Jalali, Wohed, and Ouyang note that aspect oriented modelling frameworks often do not explicitly model the precedence of aspect application [JWO12]. They address this limitation by defining a mechanism to be used in capturing multiple concerns as aspects, where the invocation of advice must follow a certain precedence. The aim of the work is not to propose tooling around the precedence of aspect application so much as to contribute to aspect oriented design theory, providing a notation for precedence which is broadly applicable. The precedence model is, put simply, that a mapping exists for each application of advice to join point such that the mapping defines an ordering on advice for that join point. The definition defines "AOBPMN", a formalised dialect of BPMN supporting aspect orientation with precedence. A case study is provided where AOBPMN is instantiated within a coloured Petri net. Their study expands on existing work by research teams led by Capelli [Cap+09; Sil+20] and by Charfi [CM07], in that it develops a mature formalism for and model of aspect orientation as applied to business process modelling. However, Jalali, Wohed, and Ouyang note that their case study is limited in scale. No tooling or evaluation of the practical benefit of their approach is provided.

2.4 Process Variance in Simulation & Data Generation

Brief intro of the section here Brief intro of the process variance section here

Typically, a simulation concerns a single process. This means that all expected behaviour must be included within that process; complex or contingent behaviour must be represented within it. The techniques reviewed here offer separately including some modification of a process (or represent the modifications of varied process within the simulated output [SA13; SA14]). The benefit of this approach is that possible changes to a process can be described once and applied to that process where appropriate. Process changes might describe attempts to circumvent security protocols, laziness or confusion in a human actor within the model, or random "noise" so as to produce synthetic log traces containing aberrations which mimic those found in noisy empirical datasets. In all cases, behavioural variations can be described as some alteration to a process and applied to either a model or the product of that model (datasets or log traces) to represent the same alteration introduced at an arbitrary point of the simulation.

This decouples the expected behaviour in the original model from simulated behaviour, which is obtained by composing the model and behavioural variation using a given technique's method for doing so. This approach to modelling behavioural variation allows the same altered behaviour, which would otherwise be described in many disparate points in a model, to instead be written once and introduced wherever required. The observation that the same variation might appear in many areas of a model, and that the variation can be separated from the model and introduced where necessary, frames the modelling of these variations in the same way as aspect orientation frames cross-cutting concerns. The work presented in this thesis explicitly applies changes to processes and simulated behaviour as aspects in the same manner. Therefore, although this aspectual connection is not made explicit in much of the literature to date, it is important to review literature on simulation and modelling which modularises these variations; this section reviews that literature. The work reviewed is highly relevant to the contributions in this thesis, in particular because the core motivations of this field are shared by this thesis; the section therefore leads with a subsection discussing those motivations, and their relationship to aspect orientation, in detail.

2.4.1 Discussion of Variation & Motivations for Variations in Process Models

Mitsyuk et al. are motivated by the field of process mining's requirement for datasets of process logs made from well-understood process models, defined in a high-level manner [Mit+17]. They demonstrate a technique for generating event logs from BPMN models by introducing algorithms for the direct simulation of BPMN models and the collection of traces from those simulations. While their approach does not support the simulation of all BPMN concepts, notably message passing, they provide a tool which produces log traces for a BPMN model through PROM, a standard tool within the process mining community [Van+05]. This results in their technique providing high-level model simulation through already-standard tooling, meaning adopters of the technique need not rely on dedicated tooling which may not be compatible with other researchers' process mining techniques.

The algorithms presented by Mitsyuk et al. simulate processes described by BPMN models, but don't include any provisions for representing variance. However, the technique could plausibly be combined with aspect orientation techniques for BPMN as discussed in **??** [CMM10; Cap+09] to represent alternate behaviour applied contingently. Demonstrating the viability of this approach is an avenue of research beyond the scope of this thesis. However, the motivation of the work mirrors that of other research projects reviewed in this section: a need for synthetic datasets of traces through a process, for use in scenarios where empirical datasets are difficult to obtain.

Difficulties arise when obtaining real-world datasets for many reasons. For example, large empirical datasets are typically produced by organisations which would prefer some level of secrecy around their operations, making publishing those operations for the investigation of research teams unlikely. Researchers collecting these datasets describe a "lengthy process" [Don20] and explain that traces of real-world processes are hard to obtain because "higher management [can be] worried about the risks" of publishing such datasets [Don20]. Another factor contributing to the difficulty of collecting empirical datasets is that they often cannot be collected, either because there is a need to study the process before implementing it (making synthetic datasets the only option available to researchers Find citation for empirical datasets being the only ones available to researchers, maybe for disaster prevention or similar?), because the process is not yet fully understood (making simulation of many variants of that process useful in aiding understanding find a citation for simulating different systems for finding an optimal one. Arguably Genetic programming & hill climbing do similar things?), or because the dataset itself is of use to researchers, not the real-world system that produced it (such as in the case of evaluating process mining techniques [VW04; AGL98]). Find some nice way to round this subsection out, or refactor out subsections Important to acknowledge somewhere in this subsection that synthetic data generation is a well-researched field, but that generating logs from simulations with variance is the specific area relevant to this thesis

2.4.2 Representing Variations in Process Models and their Outputs

Research undertaken by Stocker and Accorsi [SA13] aims to synthesise process logs which are representative of attackers' efforts to compromise the security of a modelled system. Their research project, named "SecSY", is an attempt to address issues arising from the difficulty of retrieving representative log traces for security-critical systems in which attacker activity is present. Logs are developed by process simulation through "well-structured" models, a mathematical property on which transformations were previously defined by Vanhatalo, Völzer, and Koehler [VVK09]. The authors develop a tool for the simulation of a process using well-structured process models, and apply transformations to both the model before execution and the log it produces through the trace of a simulation. They conclude that their tool is performant, and verify it can produce logs representing security violations by way of analysis through PROM, a popular framework for process mining, and pre-defined security constraints on their models. They note that log traces cannot be interleaved (due to a lack of parallel simulation of processes), may be incomplete (missing violations), and that mutated models and traces are not guaranteed to be sound by construction. However, they see their proposal as a necessary step in realistic data generation for business processes. A weakness of the work is that model and trace modifications are relatively rudimentary: processes can be added or removed, but complex graph transformations are presumably only permissible when representable through the composition of the mutation primitives they provide, on which there are only three for processes: swapping `AND` and `XOR` definitions of process gateways, and swapping process order. Mutations cannot be applied contingent on the state of a simulation run, for example, representing a decision taken by an attacker based on what had already happened. In later work, Stocker and Accorsi detail the technical aspects of SecSY, their tool for implementing the generation of synthetic logs which use their technique [SA13] to represent security violations security-critical business processes. A Java implementation of SecSY is described, which simulates well-structured models and applies mathematically-defined transformations on the model being simulated (before simulation occurs) and the logs obtained through simulation traces. An improvement on earlier work is that custom transformers can be written. However, a limitation of the original work remains, which is that users cannot easily dictate the degree to which variations are applied.

Pourmasoumi et al. also address the need for access to variations on business processes, though for the development of a research field, "cross-organisational process mining" [Pou+15]. Process mining can require many process logs, as does the benchmarking and evaluation of process mining techniques. Traces from business processes which are similar but not identical can produce log traces which reflect that similarity, but also reflect the variations in different instances of those processes. These log traces exhibiting variation can be used in the training and analysis of process mining tooling and techniques, which must contend with natural variation present in the execution of real-world traces. To support the field, log trace generation from a variety of process models is therefore required. Such logs are not in adequate supply, as explained in section 2.4.1. The authors' approach to the problem is to present an algorithm for the mutation of business processes, such that simulation against variations of the business process can produce process logs reflecting those variations. Their algorithm makes use of structure tree representations of processes, which models processes as trees and permits conversion to BPMN models and Petri nets [Bui14]. Pourmasoumi et al. make use of this constraint to demonstrate that their models are block-structured, a mathematical constraint on model structure which 95% of models are shown to comply with [Li10]. Their contribution is a set of transformations on structure trees and block-structured models, and an algorithm applying these transformations to process models, and a tool which implements it built on PLG, a process log generation tool. They conclude that tools such theirs can be used to generate log traces representing process variation, in such a manner as to satisfy the requirements of the process mining research community.

Pourmasoumi et al. describe a list of transformations they explain is "not intended to be comprehensive" [Pou+15], which makes its full potential unclear. Additionally, processes their tool applies to must be block-structured. The importance of this requirement is that it limits their technique to business process models. It is not demonstrated that models of processes in other domains satisfy the condition, such as the flow of data [WC10] or behaviour of human or technical actors in socio-technical systems [WS18a]. Finally, the tool is limited by its lack of capacity to represent variations which are applied contingent on system state. A use case for modelling behavioural variance is to model changes which are impossible to anticipate from the vantage of a modeller, such as a socio-technical system's human actors' mistakes, security exceptions and violations, or corrective actions to mitigate undesirable system state. Pourmasoumi et al.'s tool produces variations on a process model, but modelling behaviour which is expected to vary within iterations of the *same* model is outwith the scope of their project. Shugurov & Mitsyuk's work should follow this. REVIEW IT.

Machado et al. note that there are operational costs to the inefficient modelling of business processes. Specifically, they note that processes can be replicated across automated processes, and failure to identify such scenarios give rise to these operational costs. This work's core motivation is that the representation of variation in process models would allow for the capture of a replicated process once, with instances of similar processes described as deviations from that captured blueprint. On this basis, Machado et al. extend BPMN to support the notion of individual processes as transformations of an underlying process, i.e. that a given process can sometimes be expressed as a deviation from some pattern, and is therefore define-able as the composition of that pattern and a variation upon it. Their approach is illustrated through two small, broadly similar business processes initially modelled in BPMN and then represented in Haskell, allowing the authors to demonstrate their representation of variability as process deviation with realistic examples. While the work presented makes no empirical evaluation of their technique, Machado et al. note that their industrial partner responded positively to the research presented in this publication and that further technical improvements are to be made (support for around advice, and for quantification). They also express an intent to conduct a real-world evaluation in the HR domain. While we are unaware of any real-world evaluation undertaken by this research time to date, some formal proofs that the transformations their tool supports are always well-formed [MAG12].

2.5 Discussion

Aspect orientated programming is designed to permit highly modular software engineering in scenarios where cross-cutting concerns are identified, by isolating them as separate modules [Kic+97]. The aim in employing aspect oriented programming is to reduce tangling, where a cross-cutting concern is intertwined within a program's main concern, and scattering, where the same cross-cutting concern is re-written at many points in a program's source. Aspects which modularise that concern can be written once, separately from a program's main concern, and re-introduced to each point in a program to which the aspect relates by way of weaving. An aspect-orientation framework must therefore be able to quantify the points at which the aspect applies [FFN00]. Aspects specify both advice (the implementation of its associated cross-cutting concern) and the join point defining where in a target program that advice should be woven. Aspect orientation therefore implies that the source aspects are applied to are oblivious to their application [FFN00].

In theory, the design of the paradigm is such that it should be expected to increase modularity in the software applying it [Kic+97; FFN00] more citations , and its proponents often claim this modularity as a benefit of the aspect-oriented approaches of their research [GS04; CM07; Cap+09; JWO12; CBB13]. However, its critics question the reasoning around these benefits, and note that there is little empirical study into whether aspect oriented programs truly benefit as a result of this modularity [CSS04; Ste06; Prz10].

One appropriate application area may be in the representation of behavioural variation in simulation and modelling. The application of aspects to models is already well-studied [ABZ08; CBB13], particularly within aspect-oriented business process modelling [CM07; Cap+09; JWO12], where modelling behavioural variation has also seen some prior research [Mac+11; SA13; Pou+15; Mit+17]. Outside of business process modelling, aspect orientation would reasonably be expected to support the development and observation of models themselves [GK99].

Research opportunities at the intersection of aspect orientation and the modelling behavioural variation occur because behavioural variation is an example of a cross-cutting concern. Changes to expected behaviour such as laziness, boredom, confusion or misunderstanding can impact many parts of a process in a socio-technical system, but modelling the variation caused by any one of these requires similar alterations to behaviour in many disparate parts of the model they occur within. Behavioural variations are therefore both scattered and tangled, and constitute a cross-cutting concern which might be well suited to modularisation in aspects.

Existing aspect orientation techniques and behavioural variation modelling techniques are ill-equipped to take advantage of this alignment. That behaviour changes when it varies is tautological; however, changes supported by existing aspect orientation techniques weave advice before, after, or around their join points, and therefore do not alter the definition itself. In some systems, some variations may be representable as additions inserted before and/or after some other behaviours, but such techniques are unsuitable for representing modifications of the target behaviour itself, or behaviours which should be omitted instead of added. Additionally, join points available to an aspectual programmer may not be granular enough to permit representing the changes they require in such as system, and aspect orientation's principle of obliviousness opposes the modification of target code to make new join points available. Techniques which would directly rewrite target source are typically extremely low-level, and therefore ill-suited to most modelling applications [KH98]. Other techniques which permit defining changes to processes at a high level may allow a modeller to *describe* intended changes (such as the high-level annotations supported in AOPML [Cap+09]), but such techniques are intended for human interpretation, not machine execution for simulation purposes. These techniques permit describing behavioural variation within another process, but only by virtue of the flexibility of natural-language annotation, and are therefore unsuitable for simulation and modelling purposes.

Such techniques also lack executable notion of "state". Real-world behavioural variance can often be contingent on the environment an actor exists within. While variations such as security breaches might be predictable (by identifying weaknesses in existing processes, for example), variance in socio-technical systems often occurs in the behaviour of human actors. This might be in response to a degraded mode [JS07], where behaviour naturally drifts to a functional — but undesirable — state, or due to an individual making a mistake, forgetting procedure, or being in a state which alters their behaviour, such as tiredness or drunkenness [OLe20]. A framework for modelling behavioural variation using aspects should therefore apply that aspect to a simulated system contingent on the state of that system at a given point in time. High-level modelling technologies such as BPMN and OPM are executable [Mit+17; DOR95], but it is not trivially evident that executable versions of these technologies are compatible with aspect-oriented modifications of their modelling language [CMM10; Cap+09]. As noted, low-level program transformation technologies are also unsuitable. Techniques for applying variations to models exist, but are unsuitable for simulation (and therefore cannot represent application based on temporal state) [SA13], produce individual models representing each possible instance of a variation [Pou+15], do not support the dynamic weaving of aspects for contingent behaviour [Mac+11], or attempt to represent the changes one would expect in the output of a simulation by executing an unmodified simulation and amending its output directly [SM14]. None of these techniques are suitable for representing a behavioural change which is applied contingent on state. Incidentally, these techniques for modelling behavioural variation also lack support for the alteration of a process definition, or changes "inside" a process definition, as discussed earlier, which also makes them unsuitable for simulated behavioural variation.

Should a tool exist which dynamically weaves definitions of behavioural variations for contingent application and which is capable of expressing changes within a join point rather than before or after it, a challenge would arise as to demonstrating the benefits the tool achieves. Contingent application of behavioural variation, and the ability to define changes to processes specifically, would fulfil the opportunity in marrying aspect-orientation and modelling behavioural variation, but the benefit offered by such a tool is unclear. The introduction of oblivious modification to a model may break its representation of its real-world analogue, making such a model difficult to reason about. Added to this is the complexity of the tool's capacity to rewrite any join point's definition. Though aspect oriented literature often lacks case studies demonstrating the benefits of the approach, it is particularly important to investigate whether such a tool could produce realistic models, and whether the expected benefits of aspect orientation as applied to the model hold in practice. In particular, would engineers benefit from the modularity afforded by isolating behavioural variation into an aspect, and would the resulting aspectual modules be re-usable when modelling other systems?

2.5.1 Research Questions

Not sure exactly how to write this. Eeep!

Chapter 3

Prior Work

An implementation of the main tool developed and used in this thesis, “PyDySoFu”, predates this thesis. As context for the contributions in this thesis, this chapter will describe the state of the project before the presented research was undertaken. Motivations for the tool’s original development are described in section 3.1, which are followed by its design and implementation in section 3.2, and that of related tooling for experiments and simulations in section 3.3. The chapter concludes with a description of the research undertaken using these tools in section 3.6, as some results in the representation of behavioural variance using aspect orientation were found using these tools which predate this thesis and offer important background for the research undertaken in it.

3.1 Motivations in originally implementing PyDySoFu

The original incarnation of PyDySoFu was developed with different motivations than those outlined in chapter 2. It is therefore important to elucidate the context in which it was designed and developed.

PyDySoFu was originally developed for the representation of behavioural variance in sociotechnical systems, and was first produced as a proof-of-concept; the core contribution was one of tooling. It was developed for use in Python codebases, by virtue of the language’s widespread use and its flexibility in its modelling of data and process.

The original version of the tool was to be applied to models of behaviour in socio-technical systems, where individual actions were represented as functions. Actions which could be decomposed further into more granular actions were to be defined as functions with calls to invoke their more granular counterpart functions. Invocations of low-level behaviours would implement some change to an environment in the model which its modelled behaviour would be expected to incur. Invocations of high-level behaviours, containing the invocations of lower-level behaviours they compose in the model, would therefore apply the combined effect of the collected behaviours they represent. A benefit of this approach to modelling behaviour was that high-level behaviours could implement the “flow” of a behaviour. For example, a behaviour which would be modelled in a flowchart as having some loop could be modelled analogously in the method described through use of primitive control flow operators in Python, such as `for` and `while` loops.

Another benefit of this approach is that the behaviours modelled have a predictable structure which is amenable to metaprogramming. A low-level behaviour's affect could be changed by changing the function definition; more structural changes could be made by altering the flow of less granular behaviours. A simple high-level behaviour containing a series of function invocations (modelling an ordered list of steps in the socio-technical system) can be represented as a literal list of function calls. The contents of such a list is trivially modifiable. Removing an item from a list or truncating it at a certain length, for example, are both achievable in a trivial manner using high-level languages such as Python. Notably, many behaviours can be conceived of which could be represented as high-level behaviours but would not be amenable to a simple list of more granular behaviours, such as a behaviour with a looping quality.

With a mechanism to rewrite either an implementation of a behaviour or a collection of behaviours (in the less granular functions mentioned), modelling in such a fashion could therefore lend itself to semantically simple metaprogramming that could represent real-world variations in behaviour. However, largely for reasons discussed in **??**, metaprogramming for representing realistic behavioural variations in socio-technical simulations should be able to take advantage of system state. Many real-world behaviours are contingent on environmental state. Real-world actors in socio-technical systems might become tired after lots of work, or proportionally to time of day within a simulation. Therefore, the metaprogramming as described should be performed during runtime, for which no suitable candidate was available. PyDySoFu was developed to fulfil this requirement, so that behavioural variance in socio-technical simulation could be modelled as described and subsequently studied.

3.2 Original PyDySoFu Implementation

To disambiguate the improvements made to the original PyDySoFu implementation in the tooling contributions of this thesis — and to explain the fundamental concepts involved in both implementations' approaches to aspect orientation and the application of behavioural variance — the implementation of the original PyDySoFu tool, which predates the work presented in this thesis, is described here.

3.2.1 Weaving mechanism

The original PyDySoFu implementation made use of a weaving mechanism which could be categorised as "total weaving" in the parlance of Chitchyan and Sommerville [CS04]: hooks to apply advice are woven into every possible join point. The library was implemented in Python, which offers a flexible object model PyDySoFu is able to take advantage of in order to weave its hooks.

Python's object model has three key properties which the original implementation of PyDySoFu takes advantage of:

① Everything in Python is an object, including types, functions, and classes

② Objects are, in essence, implemented as a dictionary[1] with string keys. All attributes of an object — such as a method on an instance of a class — are values in this dictionary, and their identifiers are the string keys of the dictionary. In essence, `someObject.val` is notionally equivalent to `someObject.__dict__['val']`, though the subtleties of this mechanism will be explained later.

③ Operations on objects are handled by "magic methods", which are specifically named methods on objects which Python calls for fundamental behaviour in the language. For example, `objA ==`

[1] Python's name for what other languages might call a `map` or `hashmap`.

objB is interpreted by Python as `objA.__eq__(objB)`. Other built-in behaviours in Python have similar reserved method names which represent the implementation of some behaviour. These methods can be overridden, or specified by a programmer if they wish.

PyDySoFu weaves aspect hooks into classes by taking advantage of these three properties of Python. At a high level, PyDySoFu operates by replacing the `__getattribute__()` method of a class object with a custom one. `__getattribute__()` is the magic method responsible for performing lookups in an object's underlying dictionary. The replacement `__getattribute__()` also looks up attributes in the relevant object's underlying dictionary, but it also searches for advice to be applied when performing these lookups, and applies any advice it finds. The replacement `__getattribute__()` method represents the aspect hooks woven by PyDySoFu.

Hooks are applied to a class by way of an invocation to a function, `fuzz_clazz`, which takes a class as a parameter and weaves aspect hooks into that class. Add citation for pdsf, asp `fuzz_clazz` replaces the `__getattribute__()` method of the class with a new function object which it constructs. The replacement function object operates

More specifically, the replacement `__getattribute__()` method makes a call to the class' original `__getattribute__()` method to retrieve an attribute when required. If this attribute is not a function or method, it is returned by the woven `__getattribute__()` function and the program affected class behaves as if it was never altered. If an attribute to be retrieved is a method or function, however, a new function is constructed to be returned. This function

3.2.2 Applying Process Mutations

3.3 Additional Simulation Machinery

3.4 FuzziMoss

3.5 Theatre_AG

3.6 Using PyDySoFu for sociotechnical simulation

Chapter 4

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…

4.1 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 [CS04]) and the use previously found for PyDySoFu (see [WS18a; WS18b]), 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 [CS04], 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 [Kic+01], 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.

4.2 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*.

4.2.1 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 [con21]. 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,

[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.

and attempts 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.

4.2.2 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.

4.2.3 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:

```
with AspectHooks():
    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[VD09], although more focused guides exist in the Python community [con16].

```python
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 4.1: 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. 4.1.

4.2.4 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 [CS04] 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

[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 (on the 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 [Prz10; CSS04; Ste06]. 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 [Cla73; CSS04] 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. 4.1 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.

4.2.5 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

4.3 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 [Tha+17].

Chapter 5

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[KM20] 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 play The 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 chapters cref the chapters on specific experiments at the beginning of the RPGLite chapter. .

Add an outline of the RPGLite chapter here.

5.1 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 optimally Cite 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 [**empty citation**] 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.

5.1.1 RPGLite's Design Implications

The state space of RPGLite makes it unusually well-suited to analysis through formal methods. Because of this, a the datasets produced through simulation of RPGLite can be compared against two other datasets: one of real-world play, and another of what can be mathematically shown to be "correct" player behaviour.

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.[2]. The entropy of a game's state is therefore $log_2(w \times x \times y \times z \times 3 \times 2)$, where the multiplications by 3 and 2 represent the stunnedness indicator for the current player (either character, or neither), and the player whose turn it is to play, respectively. The maximum health for a character is 10 hit points, which makes the maximum entropy of a game state $log_2(10^4 \times 3 \times 2\}) = log_2(60000) \approx 16.87 bits$. Each player picks 2 of 10 characters, and can choose to attack either opponent character with either of their own, or skip their turn, for a maximum of 6 possible actions. We can therefore see that the total entropy of the entire RPGLite game is no greater than $log_2(10^4 \times 6 \times 2 \times \binom{10}{2}) \times 6 \approx 24.36 bits$.[3]

Iterating through these states allows us to map the entire state space of RPGLite. As the state space the game defines is relatively small — for comparison, mapping valid positions in chess takes about 136 bits[Nie77], and this figure does not account for valid *moves* within the game, which our calculation for RPGLite does — it is feasible to analyse every possible game state. Note that the figure of $\approx 24.36 bits$ includes movements *between* states as well as the states themselves, meaning that it is feasible to map

[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 be stunned at any time, and the status effect immediately resets, meaning there are only three possible states for stunnedness: either character belonging to the player taking a turn, or neither.

[3] The actual figure is smaller:

- Players may choose two characters with special abilities that prevent them from attacking both opponent characters at once (this accounts for 9 out of 10 characters), giving 5 possible actions in a turn, rather than 6.
- The "metagame", which refers to the percieved "best" strategies at any given point in time, would impact the chance of a player selecting making certain moves or choosing to play with certain characters. RPGLite is designed to be slightly "unbalanced" in the parameters of different characters such as health, attack damage, or potency of special abilities, meaning long-term players are expected to learn effective playstyles and adjust accordingly.

Players' behaviour is therefore less uniform than this calculation would imply, but the calculation provides a *maximum* entropy of the game.

the potential progressions through all possible games of RPGLite using formal methods. Further, this allows us to understand the chances of a given player winning given transitions between different states, for example by representing moves in the game as transitions in a decision diagram where nodes are the game's state. We can calculate exactly how "good" a move is, by comparing chances of success making a given move in a given state against the chances of success when making the calculably-optimal move.

In this way, RPGLite's design allows it to be understood formally, yet it also draws on common game design elements and is sufficiently "interesting" to generate data from a real-world playerbase.[4] This yields some properties that are interesting for the purposes of aspect-oriented simulation:

① Simulated moves can be selected naively, i.e. at random, but can also be made perfectly according to the known-correct move in a given game state, or made with some calculated "cost" as to the chance of winning.

② Real-world players' behaviour can also be analysed according to the same metrics: for example, moves made by players can be analysed to understand bias, whether players learned to play "better" moves over time, or whether they selected known-strong characters more frequently than those who can be formally shown to have a relatively low chance of winning games.

③ As the actions taken when playing RPGLite are consistent — such as deciding the target character of an attack, or a character to use in an attack — random play can be simulated as a "naive" play style, which can be compared against real-world players. Where player behaviour does not correlate to naive play,[5] the biases of players may be represented as aspects which are applied only to specific actors within the simulation.

④ Should aspects be suitable as a manner of accurately representing biased play, aspects offer a separation of concerns within the simulation: any nuance found within the playstyle of specific real-world players would be replicated and applied not to the model itself, but to specific simulated players. Playstyles might also be mixed with the application of multiple aspects.

Whether aspect orientation is suitable for the realistic simulation of RPGLite gameplay is the topic of

[4] Data collected from several thousand completed games can be found at [KWM20].
[5] The concept of play correlation is introduced in chapter 6.

the remainder of this thesis. However, the design of RPGLite allows for a controlled system where a clear notion of "good" and "naive" behaviours can be defined, the system is closed insofar as all interactions within the system are known and all game elements are precicely understood, and all interactions take place between experiment participants for data collection purposes, allowing for a large dataset to be collected without information being removed due to players not consenting to their data being collected and disseminated for science. In short, RPGLite's design constitutes a system where all aspects are well-understood, no interference is anticipated from system components which are unknown or outside of experimental control, and lots of data can be collected for analysis. Therefore, the data generated by gameplay is suitable for comparison against datasets outputted through aspect-oriented simulation. This can be used to assess whether simulated players with aspect-affected behaviour accurately reflect the playerbase, and so can help to assess whether aspect orientation is suitable for realistic simulation.

5.2 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, a mobile online multiplayer version of the game was developed for data collection purposes. 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 to simulate real-world player behaviour.

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[Wal+21], might produce a more complete overview of the development.

5.2.1 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[6]. Most assets

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

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...?

5.2.2 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.

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?!

5.3 Empirical Play and Data Collected

In total, players produced a dataset used in this PhD comprising around 4,000 games Find the exact number of games analysed entirely completed[7]. 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[8]

[7]"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.

[8]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 6

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 thesis Cite 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.

6.1 Aims

Aspect orientation's use in previous simulation and modelling efforts have typically focused on the use of aspects to compose model or simulation details There 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[JS07].

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[WS18a]. 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[SW16]. 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[Kic+01] or PROSE[PGA02]) 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.

6.1.1 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[Hil+00], and similar additional complexity is added in seemingly every alternative framework, through the use of bespoke virtual machines, compilers, translators, or languages[**rajan2006nu_towardsAO_invocation**; PAG03; SL07; BH02].

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).

6.1.2 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". Furthermore, changes to a model could alter its representation so as to weaken its mimicry of the system it simulates; adding behaviours could make it *less* realistic. The fundamental issue at play is that it is unclear whether the changes made would properly represent what might be empirically observed. 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. It is therefore important to demonstrate the efficacy of PyDySoFu and the

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

modelling paradigm it introduces, by confirming the realism of a model to which behavioural variation is applied.

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 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, a naive model of play is produced. Aspects are developed which model learning within the system defined by the naive system. The synthetic datasets produced by models with naive or aspect-applied models can be compared to an empirical dataset sourced from real-world players of the game, and their similarity compared.

"Naive" is used here to describe a model which does not encode any understanding of the players of the game being modelled. Traits such as learning, distraction, or aptitude for similar games are irrelevant to the naive model. We need a naive model to demonstrate the effectiveness of an aspect-oriented alternative: we can measure how closely it reflects empirical data, and compare this against the same measurement drawn from another model with behavioural variations encoded using aspects. A closer match from our aspect-oriented model would demonstrate that the technique can enhance a model's realism. Our naive model also provides a null hypothesis: if no improved similarity is observed, the technique brought no improvement to the model's realism. No measurable difference indicates that weaving behavioural variations as aspects has no impact on a model's realism.

Write a little here on the learning aspects.

Flesh this out as a brief wrap-up of our experimental technique. The end of our "naive" model explanation might be useful to move down here / rework into this para. Contrasting the similarity of the empirical dataset to both naive and aspect-applied datasets… This discussion is provided so as to provide context for the following sections; experimental design is discussed in more depth in section 6.5.

6.2 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 information pertinent to a simulation in an applied aspect, the model was written so as to contain 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 that a function invocation (or "step") represents 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[Mat08]:

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 receives these three arguments at a minimum. Because steps of the model are functions, and therefore valid join points, aspects applied to these 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. A graphical representation is provided in fig. 6.1. Position this figure properly on the page. Maybe reword? Pretty verbose. The model it describes produces synthetic data. It does so by simulating players interacting with each other in a broader RPGLite ecosystem. The large number of players is important: future behavioural variations introduce behaviours which guide future actions based on past experiences. Non-determinism early in the model can therefore cause different players to act in different ways. Simulating many players allows the system to represent a wide variety of early experiences, as would occur to real-world players, too. As the aspects being woven simulate learning, early experiences which cause players to learn maladaptive strategies — or simply strategies less optimal than those learned by their peers — should later contrast against winning strategies, so that players with less useful early inferences can learn from players who play more optimally.

Consider changing fig. 6.1 to only include details about the model of the game itself, leaving the steps about data analysis to the later aspect-applied version (which is the real experiment). Move

[2]This is different to environments in some other simulation frameworks, such as SimPy[dev21], 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.

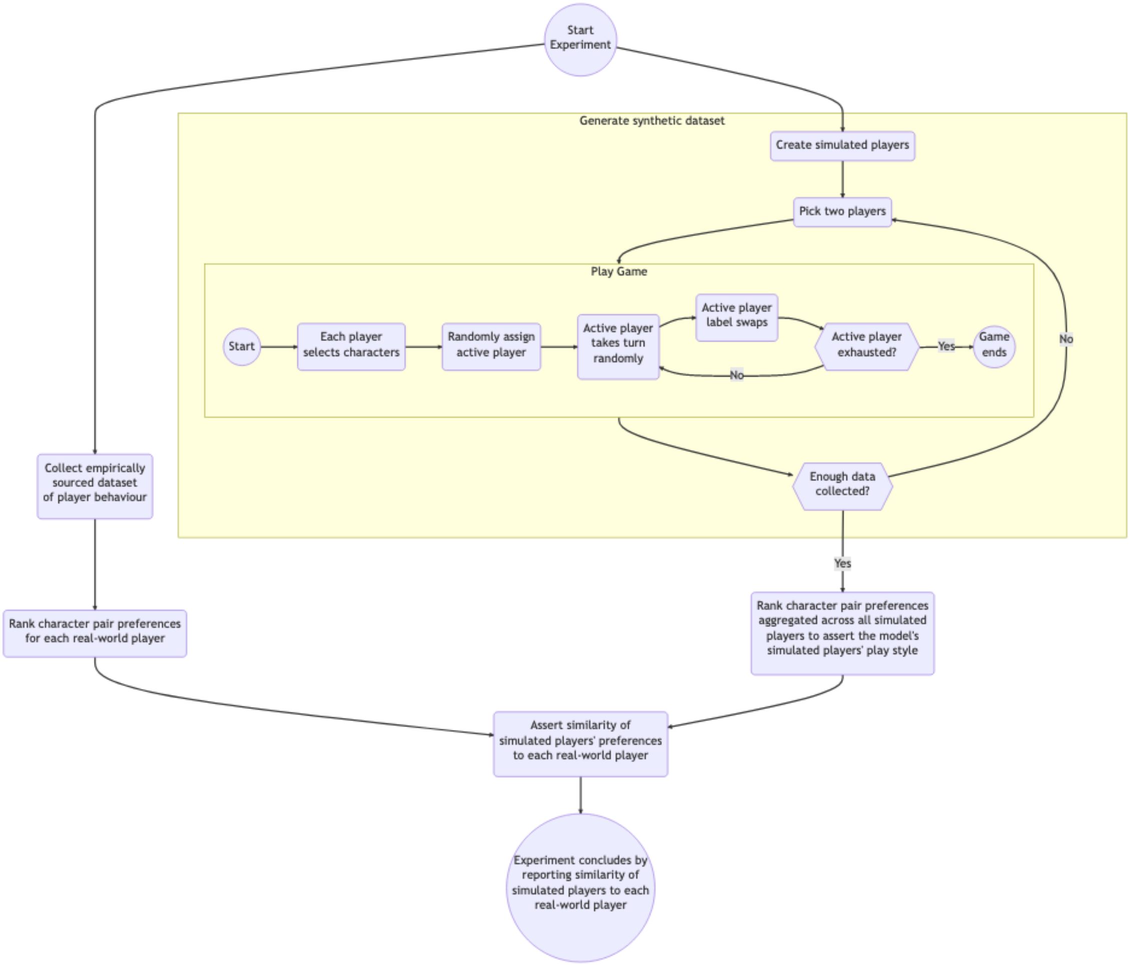

Figure 6.1: A flowchart diagramming a simple model of RPGLite play, without aspects applied.

fig. 6.1 to .svg for fidelity The stages of the naive model as laid out map to those encountered by real-world players. Two randomly-selected players repeatedly select characters to play (from the pool of 8 characters available in the real-world game), and a player is chosen to play first at random. That player selects a random valid move to make.[3] The active player alternates, and the process repeats, until such time as an active player starts their turn with both of their characters fully depleted of health. The player with remaining characters is the victor, and another game is started by picking random players and starting a game between them, until a predetermined number of games has been played.

After a sufficient number of games are played, analysis of the various datasets collected can begin, although this is discussed in more detail in section 6.5.

6.3 Experimental Design

The naive model above describes a simulation of RPGLite itself; however, the models exists to support investigation into research questions, and cannot do this alone. To answer our research questions, the model must be augmented so as to represent real-world play. This section describes the modifications made to the naive model (through aspects applied by PyDySoFu) and how data from those simulations can be analysed to arrive at answers to our research questions.

The questions these simulations seek to answer are:

① Can we fit model details per-player to get realistic player behaviour?

② Can we cluster based on accuracy of different models for each player?

③ Can we generate predictive data for unknown models from known ones?

These RQs are taken from an early outline of a thesis structure. Maybe they need rewording?

To investigate these, we produce models of learning which can be applied to the naive model to augment player behaviour. Simulated players will have their behaviour augmented in relation to the character pairs they choose to play. Real-world players could reasonably be expected to choose

[3] Many simulated decisions are random; this is because the model is designed to be naive, so it avoids informed decisions where possible. Informed decisions are expected to be woven later as aspects.

character pairs which are stronger together over time. In addition, it is possible to calculate objectively optimal character pair selections in RPGLite Cite William's thesis as a source for objectively optimal character pair selections. ; therefore, players should be expected to gravitate toward more optimal character pairs and away from poorer choices as they become more experienced in the game.

A few facets of gameplay are expected to be learned by players over time, and would theoretically also be suitable objects of simulated players' learning. One analog to character pair selection could be move selection: players would be expected to make increasingly optimal moves as they play. However, data can be muddied by players understanding the optimal use of some characters' moves better than others, as different characters invite different styles of play. Small changes to game states (the opposing team's composition, amount of health remaining, opponent play styles) could influence player behaviour also. Fundamentally, the state space of moves to be made and scenarios in which to make those moves is extremely large, making simulation computationally expensive. By comparison, character selection happens before gameplay starts, eliminating in-game factors which might influence player decisions.

Modelling the learning of character pair selection allows for modelling of players learning the metagame, rather than the game itself. Modelling learning of the metagame allows us to model players learning about the game and other players' interactions with it, rather than learning the game itself. An example of the distinction is the difference between learning how optimally make moves with a Barbarian in a team, and learning the mistakes other players commonly make when playing a Barbarian to exploit their weakness. As RPGLite is "solvable" — there is a provably correct way to play the game — players picking sub-optimal character pairs can be expected to observe that they are frequently beaten by better ones, increasingly favour pairings they identify as successful, and so converge on strong character pairs. Convergence en-masse would indicate a stable metagame. This was observed in practice in real-world RPGLite play [KM21]. However, while players made fewer mistakes over time when playing RPGLite, the average cost of a move to their probability of winning increased after many games (though decreased after a few). [KM21]. Player behaviour which converges on provably optimal play is simpler to represent than players' initial improvement followed by a seeming loss of performance. RPGLite Character selection is therefore a more convenient metric to model players' interaction with than move selection, as it presents a smaller state space for simulated players to explore while also providing a simpler expected end state for those simulated players to converge to.

Models of learning are therefore applied to players' character selections. Our primary goal in applying models of learning is to demonstrate whether players can be accurately modelled by augmenting a naive model to represent them in particular, thereby answering the first research question. Different real-world players are expected to exhibit different styles of learning; thus, multiple models of learning will be applied. Different players might also learn differently, but in the same style; for example, two players might exhibit similar biases, but one could be quicker to learn from experience or another more cautious in the application of new knowledge.

To identify models of learning and parameters for those models which most realistically represent real-world players, we look to optimise the parameters for each model, for each player, by running multiple simulations of players learning in a particular manner (with particular parameters), identifying their preferred character pairings, and calculating the similarity of the simulated character pairings to that of a player in the real world. By doing so, we can anneal to a parameter which represents the optimally realistic model of a player learning in a given style. If the data produced by this optimally parameterised model of learning is not similar to empirically sourced data with statistical significance, we determine that the model of learning applied does not represent how a player learned in the real world: even the closest dataset the model produced fit its target dataset poorly. However, if a parameter can be found for which the model reliably produces character pair preferences which align with the real-world players' with statistical significance, then the model can be said to realistically represent that player's learning of the character pair metagame.

This section will first discuss models of learning generally in section 6.3.1, and will explain how aspects implementing these models can be produced in section 6.4. The strategy used to implement these models of learning is discussed in section 6.3.2. The design of experiments investigating the aforementioned research questions by applying these aspects is then described in section 6.5. Rearrange either this paragraph of the sections it refers to so they're in order / make sense. At time of writing they refer to 6.3.1, then 6.4, then 6.3.2, then 6.5…!

6.3.1 Models of Learning

Different people learn in different ways. Indeed, no universally-accepted definition of learning appears to exist. This is presumably because it is convenient to define what it means to learn differently in the context of different pieces of work. Cognitive models of learning can be useful when considering mental processes specifically, for example, whereas functional models of learning could lend a more empirically applicable perspective. What it means to learn is clearly outwith the scope of this thesis; however, our experiments will include models of learning. To justify our model, we consider a functional approach to learning, as considering learning in this way appears more closely linked to the empirically focused work of modelling real-world behaviour than alternatives.

Lachman identify[Lac97] that standard definitions of learning along the lines of, "Learning refers to a relatively permanent change in behavior as a result of practice or experience" have practical shortcomings such as a focus on behavioural change (as learning may not change behaviour) or conflating learning's process and its product (the process by which we learn is not obviously identical to its result, of which behavioural change is an example). They suggest learning might be better defined as:

> " (...)the process by which a relatively stable modification in stimulus-response relations is developed as a consequence of functional environmental interaction via the senses [...] rather than as a consequence of mere biological growth and development [Lac97]."

(...)

They note that their definition distinguishes learning from phenomena such as injury, changes to one's maturity, or sensory adaptation, incorporates stimulus-response relationships the research community consider as learned, and differentiates learning's process and product. Their model is inherently functional, making it useful for the purposes of simulation and modelling, although they offer only a definition of learning and a brief comparison to the standard textbook definition they introduce. The work presented is not intended to demonstrate its improved model of learning empirically, only to discuss its semantic merit. However, the models proposed in this thesis require only a theoretically informed, sound basis for their model of learning, and a lack of empirical justification is not a barrier to the relevance of the model Lachman have proposed.

De Houwer, Barnes-Holmes, and Moors propose a functional definition of learning which is primarily concerned with providing a definition of learning which is both accurate and useful for the purposes of cognitive learning research [DBM13]. Doing so attempts to provide a model around which some concensus can be reached; learning is a central concept in psychology, and they see their definition as supportive of cognitive work. They introduce their definition as follows:

> "Our definition consists of three components: (1) changes in the behavior of the organism, (2) a regularity in the environment of the organism, and (3) a causal relation between the regularity in the environment and the changes in behavior of the organism."

This model of learning contains more nuance than the "textbook definitions" of learning they paraphrase as, "a change in behavior that is due to experience", but does not stray far from the core concept: some environmental stimulus impacts behaviour in a causal fashion. Their introduction of "regularity" to their definition refers to the presence of the stimulus with some form of repetition, whether this be multiple instances of a stimulus at different times, or the same stimulus occurring concurrently. De Houwer, Barnes-Holmes, and Moors explain that such a model is straightforward without the sweeping inclusivity of the simple model mentioned earlier, and is easily verified (although, as in the work of Lachman [Lac97], empirical verification is omitted in favour of semantic analysis).

Aside from other benefits more particular to their research community, these benefits are especially useful from the perspective of modelling learning in our case. A simple, functional definition can be captured in a software model, and introduces few opportunities for misunderstanding or misapplication. It also introduces helpful concepts — such as regularity and causality — which the other definitions discussed do not. For the purposes of this thesis, we therefore adopt this definition as a basis for our model of learning.

6.3.2 Modelling Learning in RPGLite

We use De Houwer, Barnes-Holmes, and Moors's definition of learning [DBM13] to arrive at a model of learning which is encodable in aspects that can be applied to our naive model of RPGLite.

We are interested in modelling players learning a preference for character pairs over time. The model should therefore account for: how player behaviour is influenced in accordance with their learning, in line with the definition's first criterion; repetition of experience in successive games influencing the direction of a player's learning, in accordance with the definition's second criterion; and do so in a causal manner, in accordance with the definition's third criterion. We therefore look to model a causal relationship between a player's observation of successful character pairs and their future choices of character pairs.

To fulfil these requirements, a model might draw on previously successful character pairs to determine future ones. There are many ways in which this could be done. For example, we could model learning as consistently playing the character pair which most recently was observed to win a game. Any game ending naturally determines a winning pair, and we can select this pair when playing future games, until a different pair is observed to win instead. However, this does not align with our expectations around how players *would* engage with a game in the real world. Having a strategy one is confident in and being unlucky with the game's random nature is unlikely to deter a player from what they believe is ideal. Indeed, we can expect players to understand that perfect play might not be winning play: in some games, the right moves might not lead to a successful outcome due to moves randomly missing opponent characters. Equally, players may take time to become confident in a strategy; we would expect a player to explore character choices before settling on a preferred pair early in their experience, and would expect very experienced players to choose characters based on what they have learned, rather than continuing to explore options for which they have sufficient information to reason about. We can infer that:

- → There are scenarios where players can be expected to observe wins/losses without incurring behavioural change.
- → Players' confidence in what they have learned can affect their inclination to draw on what they learn when making decisions.
- → What players learn in successive games would have a small impact in their early experiences, but an increasingly significant impact proportional to their experience in the game.

The model of learning used to simulate players' improved play of RPGLite can be explained following a similar structure: a component pertaining to observations players make about winning characters; a component pertaining to their inclination to use that knowledge when choosing characters (rather than exploring their options); and a component pertaining to the strength of that inclination in proportion to their experience.

The first — that players should make observations about winning players — can be implemented by modelling the choice as a probability mass function (PMF). The PMF maps character pairs to their chance of being selected by a player, and initially tracks all character pairs as having an equal chance of being selected. After every game, the chance of selecting the winning character pair increases, and the chance of selecting any other pair decreases. The sum of probabilities of being selected across all character pairs always sums to 1 (100%). This can be implemented as a record of the winning character pairs observed by a player: many ways of producing a PMF from a sequence of wins exist, but for the purposes of explanation, one such method is to take the proportion of wins for every character pair as their probability of being selected. This method produces a valid PMF because the probabilities must sum to 1 by virtue of the winning pairs being the sole determinant of the probabilities; every win contributing to the probability of a character pair being chosen means that 100% of character pairs are considered, and therefore the probabilities inferred from that data also sum to 1.

A model of learning where prior belief as to optimal character pair is causally related to those which were previously successful fulfils the first criterion of the model of learning proposed by Lachman citelachman1997learning; however, players would be expected to explore a state space in the early stages of their experience. The model proposed by Lachman identifies that the experience of a learning agent draws from "regularity" in their environment; we therefore require that an experience affects behaviour not when it originally surfaces, but after repeated exposure to it. This fulfils the intuitive understanding of learning discussed earlier — players' confidence in what they have learned affecting their inclination to act based on experience — our second criterion. We therefore require some model of confidence an agent has in what they have learned.

The third inference — that confidence should be low initially, and grow proportionally to experience — explains the "shape" of the confidence model. We require a monotonically increasing function mapping experience, quantifiable as games played, to confidence, being a percentage chance between 0% and 100% that a player is to determine their character choice based on what was learned, rather than on an exploration of their space of possible choices.

Such a model of confidence offers a model of learning when combined with a PMF representing historical observations of a character pair's success. If the model of confidence indicates that the player is to explore their space of possible choices, they select character pairs randomly; if they are modelled as confident enough that their behaviour is instead informed by their experience of their environment, they select a character pair according to the PMF their historical observations define. As the PMF affords higher probabilities to repeatedly winning character pairs, player behaviour is causally affected by the regularity of their experience, implementing a realistic model of learning in agreement with the functional model proposed by Lachman [Lac97].

Our model of learning is therefore complete, but a model of confidence remains to be defined.

6.3.3 Modelling Player Confidence in RPGLite

To model learning, we must find some function mapping experience (quantified by a count of games played) to some probability between 0 and 1, and fitting the criteria described in section 6.3.2.

A sigmoid curve fulfils both the second and third inferences in section 6.3.2. It also notionally conforms to expectations around "confidence" as an anthropomorphic trait: much like a sigmoid, confidence starts low, and remains so until it achieves some inflection point, after which it grows rapidly until tapering asymptotically toward some maximum. However, not all real-world players might express the same traits in their growth of confidence; the shape of their sigmoidal confidence may differ. In order to account for this while guaranteeing a monotonically increasing mapping of the number of games played to a player's confidence, we select a sigmoid which can be parameterised to alter their shape.

A sigmoidal curve is suitable for this model of learning where other curves would not suffice, because we require a period where players lack confidence and explore their options. Sigmoid curves such as the logistic [Ver45] or Gompertz [Gom15] are widely used when modelling systems [WJ97], but while they fulfil the role of a monotonically increasing curve with asymptotically low and high initial and final states, the shape of such curves is not trivially modified to fit different players' learning styles.

More flexible asymptotic curves were developed by RICHARDS [RIC59] drawing on growth curves developed by Von Bertalanffy [Von38], which afford a natural pattern of growth; RICHARDS amends this curve to offer a parameterised growth rate. This curve can be made equivalent to other curves, including the logistic and Gompertz [FT+84]. This curve allows for a parameterised rate of growth, but lacks parameters controlling the points at which growth occurs most rapidly. The relative rate of confidence gain is a separate concern to the point at which such growth occurs: a player might cautiously grow in their confidence until they are already very experienced, or might bullishly grow in confidence yet plateau early, taking longer to reach complete confidence in themselves than they did to garner an initial increase, regardless of their relative growth in confidence.

The flexibility of a parameterised relative growth rate appeals to the notion that different players would gain confidence at different rates, but the point at which confidence accelerate most must also be controlled. We therefore employ the Birch curve, proposed by Birch [Bir99] for its increased flexibility as compared to the Richards curve combined with its additional parameter used to control the curve's shape. [4] Different players might exhibit different rates of growth in their confidence, and might grow maximally in their confidence at different points in their experience; the birch curve satisfies these properties as a model of confidence. Investigate: did we ever model confidence with anything other than c==1, i.e. a standard logistic curve?

6.4 Aspects Applied

Having a naive model of player interaction in RPGLite allows for the generation of a control dataset, which can be compared to aspect-applied datasets to examine whether the aspect-applied models are provably "closer" to empirically-sourced data than the naive dataset[5]. To generate the experimental dataset, the naive model producing our control is augmented through the application of aspects. These aspects fulfil different functions, and can broadly by grouped into three categories:

[4] Birch refers to shape to mean the point of inflection of a curve. The point of inflection of an exponential rise to a limit is at its initial point; the point of inflection of the logistic curve is in the exact midpoint of the curve's growth.

[5] "Closer" here refers to a similarity measurement which we will define later in this chapter.

① Aspects implementing behavioural models, changing the behaviour of simulated players;

② Aspects instrumenting the naive model to perform observations necessary for the implementation of behavioural models (such as models of learning) on the naive model;

③ Aspects altering the behaviour of modelled players to simplify experimental observation, and to handle exceptions introduced by changes within the aspect-applied model.

The first set of aspects implement the behavioural change we anticipate will produce datasets closer in similarity to empirically sourced data than that sourced by the naive model. The second set lays the foundation for applying these changes. The third applies behavioural changes which experimental observations require. Necessary changes unrelated to our models of learning will first be explained in section 6.4.1, followed by an introduction of foundational observations for our models of learning in **??**. Finally, the models of learning built on those foundations are introduced in section 6.4.2. The ordering of this explanation of different uses of aspects is confusing; once the aspects are explained individually, reorder the sections, rework the introduction to the structure of this section, and ensure the outline of the section reflects the changes made.

A diagram of a game of RPGLite with all possible aspects applied[6] is presented in fig. 6.2, by way of high-level overview of the aspects this section introduces.

6.4.1 Aspects for model improvement

Ensuring the Best Move is Played

As discussed in section 6.3, the goal within this experiment is to build a model of players' learning of character selection rather than move selection. However, randomly selected moves are liable to place players in unrealistically weak positions, as players are unlikely to make obviously poor moves such as skipping a turn with no clear reason. It would be unexpected to see true randomness in players' moves. This is a concern for the modelling of players' character selection, because the model of learning defined requires a causal relationship between what is observed (in this case characters which most

[6]Including aspects which should not be woven in the same experimental run: some aspects implement different models of learning and are therefore conceptually incompatible.

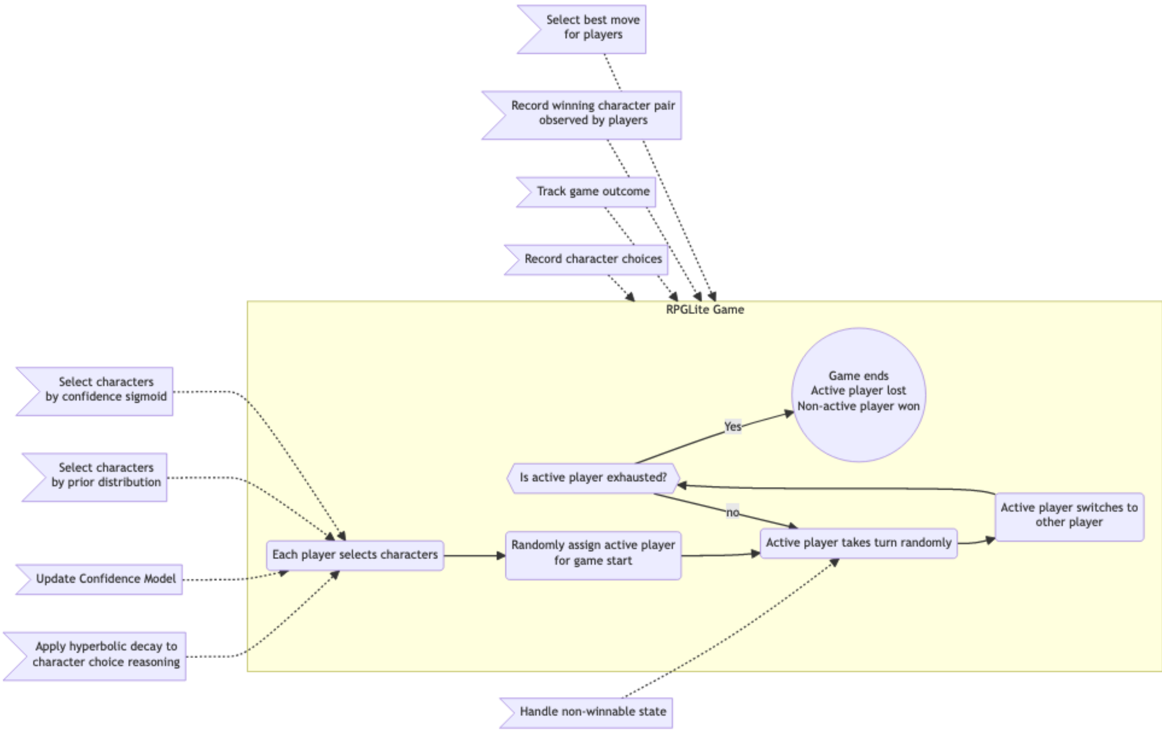

Figure 6.2: A flowchart describing a simulated game of RPGLite, and all aspects woven into the game to implement the various models of learning. Some aspects should not be woven together in the same experimental run, as they implement different models of learning.

reliably win games) and behavioural change (players choosing character pairs proportionally to their estimated chance of winning a game); if selecting random moves cause simulated players to lose games when they would have realistically won them, this would affect character selection by definition.

Character selection and move selection are therefore somewhat linked. However, move selection is most frequently optimal: in the majority of cases real-world players chose the best move available to them, as analysed by Kavanagh and Miller [KM21] :

> " (...)the majority of actions taken were optimal, with a cost of 0.0. In total 73% of the player's season 1 actions and 77.8% of their season 2 actions were optimal.[a]"
>
> [a]Different seasons of RPGLite are discussed in more depth in Make a reference to the chapter(s) where I explain different seasons of RPGLite.

(...)

Therefore, a naive model of move selection which represents real-world move choices in $\frac{3}{4}$ of cases is to select the best move at every opportunity. We therefore apply an aspect to the function responsible for move selection which performs a lookup on the dataset of action costs defined by Kavanagh and Miller [KM21], and select the known-optimal move in every case.

One additional factor in move selection is handled by this aspect. As simulated players' behaviours are augmented to select universally optimal moves, an anomaly in the dataset was identified. In some games, optimal play would result in infinite loops: players would arrive in a state where they would skip turns mutually and indefinitely. This is because it is possible that, in RPGLite, to make any move is disadvantageous. [7] Both players can concurrently exist in this state. Therefore, both players' optimal move is to skip their turn. Kavanagh and Miller note that such states were reached in 64 cases in real-world play, but real-world players never skipped their turn; therefore, this scenario is also an example where suboptimal play can be re-introduced. This aspect therefore returns random moves if the previous two moves in a game were to skip. Such a state indicates that the game reached a stable point where neither player would optimally deal damage to the other. Random moves destabilise this state and return to ordinary play.

A model of move selection more closely aligned with expectations of player behaviour is beyond

[7]This is because barbarians deal more damage once they lose a sufficient number of hit points. As a result, to deal damage to a barbarian can result in their having an opportunity to win in one move rather than two in certain health states.

the scope of this thesis: a demonstration that models of both move selection and character selection can be optimised concurrently to arrive at a realistic model of empirical play with multiple aspectually applied models of behavioural variance invites itself as future work. Do I need code snippets for notes on aspects?

Handle Game States with no Viable Moves

A consequence of playing games using only optimal moves was that unexpected states could occur. One might assume that, choosing moves optimally from a dataset of available moves in every game state it would be infeasible to arrive at states not present in this dataset. However: the dataset was produced through model checking, with the aim of identifying the cost of an action regards its impact on a player's chance of winning. If a loss is guaranteed, all moves have a 0% chance of winning; the model checker producing the dataset of available moves therefore identifies that the game is a foregone conclusion, and contains no valid moves.

In such a situation, identifying the winning player is as simple as identifying the player who made the previous move. We therefore apply an aspect which handles exceptions raised by move selection (as lookups will fail due to the game's current move state not existing in the dataset of action costs) and, after identifying that the exception indicates that the state does not exist (in this case, a `KeyError`), the aspect handles the exception by assigning the losing player's characters 0 health and swapping active players so that the simulated game proceeds with the losing player taking the following turn. As this turn starts with the active player having 0 health, the game ends as expected.

6.4.2 Aspects for Instrumentation

As explained in section 6.4, the experiment being discussed requires observational apparatus to be implemented in order to collect data about the experiment for later analysis. Ordinary experimental setups might require this apparatus to be tangled through the experimental codebase; however, aspect orientation provides a technique for augmenting the simulation to make observations before, during, or after any step of within the model. This demonstrates observational scaffolding as a cross-cutting concern as suggested by Gulyás and Kozsik [GK99]. Aspects realising this concept and supplying observations and additional models in support of our model of learning are discussed here.

Update Model of Confidence

The models of learning described in **??** have players make observations as to the character pairs which most frequently win games, and have those players alter their behaviour in response to these observations. However, as discussed in

Record Prior Distribution of Character Preferences

Looks like aspect recording prior distribution might be applied to data generation; figure out how to fit it into fig. 6.2.

Record Character Choices

Track Outcomes of Games

Record Winning Pair seen by Players on Game End

6.4.3 Aspects Implementing Models of Learning

??

Character Selection from prior distribution

Character Selection using confidence sigmoid

Character Selection exhibiting hyperbolic decay

Why didn't we make a bayesian model of learning? Should we have? Would this be difficult at all?

6.4.4 Simple Model of Learning

Describe probability updates by way of a simple PMF defined by previous outcomes

6.4.5 Modelling Bias with Hyperbolic Decay

Find citations explaining hyperbolic decay

Describe probability updates affected by a hyperbolic decay bias

6.5 Experimental Design

Specifics of how we intend to analyse our data, leading on from the brief discussion in section 6.3. Discuss specifics of how we intend to analyse data, rather than high-level experimental overview presented in earlier chapters

6.6 Experimental Results

Presentation of the results owing from the experiment as described in section 6.5, and evaluation of the research question with respect to these findings.

Again, no subsec likely required here.

6.7 Discussion

A closing discussion on what we found, and how the research question was answered.

Chapter 7

A chapter title here for the experiment moving aspects to new systems, or systems with some changes

This is a relatively short chapter; a lot of the building blocks for it exist in the previous chapter, so there's less ground to cover. If it ends up quite lop-sided, I'd chop the earlier chapter in two rather than artificially making this chapter beefier; I think it'd flow better.

7.1 (Reword) Experimental motivations / motivation of research question

This section should describe why it's interesting to move an aspect trained on some actors' behaviours to a new system, and discuss what investigating this can teach us that we *don't* already know from the previous experimental chapter. We've got aspects which represent behavioural variance. If the system changes, can we expect that these aspects still apply to the new system? Is the representation of behavioural variance in this model separable from the system the behaviour occurs within?

7.1.1 Coupling of Model and Behaviour

Explaining the issues of behaviour coupled to models. Highly related to some of the reviewed literature — I forget exactly what — which discussed whether aspects which were designed to be applied to one system could feasibly be transferred to other systems, or whether they're inherently aware of the system they're originally designed for. The core concept is that to be making changes to some underlying codebase, you probably have to know what that codebase is, in most cases at least.

7.1.2 (Reword) Gaps/opportunities left by previous experiment

In the previous experiment we demonstrated that behavioural variance can be plausibly realistic. Are those variations separable from their underlying model, or — when trained i.e. made realistic — do they suffer from the coupling discussed in the previous subsection?

7.1.3 Research Question

Whatever the specific research question's wording for this was. Something about decoupling realistic aspects from a given model maybe? It should be in an earlier chapter somewhere.

7.2 Experimental Design

We took old player data and trained aspects on them, and in the previous experiment we found they were statistically significantly accurate. We ran a second season of RPGLite with slightly different parameters on player data. We modelled how players learned and variations on their learning patterns, so in theory, we should be able to apply the new learning patterns to the other system too. Do we have to re-train the aspects? How portable are they? This section lays out the design of this experiment.

7.2.1 Changes to RPGLite

What changes did we make to our system?

7.2.2 Applying Behavioural Variations to New System

Layout of new experiment, how it'll work, what's measured, why it should answer the RQ.

7.3 Applying Aspects from a Control System to a New System

Our implementation & results from the experiment described above. Discusses how the above experiment was realised, lays out the results we found, and relates those results to the research question we started with.

7.3.1 Implementation

Implementation of the experiment

7.3.2 Results

Presentation of the results of the experiment, analysis, some discussion (more in next section too)

7.4 Discussion

Some notes discussing the outcome of this research question (that trained aspects representing behavioural variance aren't separable from the system they're trained around)

Chapter 8

Future Work

The focus of this thesis is in developing a state-of-the-art aspect-oriented framework, producing a suitably constrained experimental environment to demonstrate its effectiveness, and using that environment to investigate whether aspect orientation is suitable for simulation purposes? It's about showing that we *can* use aspect orientation appropriately in simulation environments, and that aspect orientation can also lead us to realistic and nuanced simulations, too. Go back through the pdsf and lit review chapters to make this argument properly. . As we have found that aspect orientation is appropriate in this context, successfully produced this well-constrained environment for simulation, and produced a novel aspect orientation framework which demonstrates novel and powerful weaving concepts, lots of opportunities for research outwith this thesis' scope present themselves.

This chapter describes some possibilities for the presented research to be extended in the future.

Rewrite the chunks in each future work section for this chapter, into their own subsecs with proper explorations, citations, and so on. This chapter is currently a scapbook of ideas!

8.1 Future Work pertaining to PyDySoFu

8.1.1 Aspect-Oriented Metaprogramming

The combination of metaprogramming and aspect-orientation introduces powerful new possibilities in the realm of aspect-orientation. In traditional aspect-oriented work, aspects treat their targets as black boxes. This leads to some limitations:

- Traditional aspects cannot add their behavioural modifications interspersed within the work being done by their target. The "textbook" use-case for aspect-orientation is logging: aspects can separate logging from the business logic they are applied to. However, a programmer in more mainstream programming styles may wish to insert logging behaviour *within* their business logic, rather than *around* it. Aspect-oriented metaprogramming makes this possible, as the target can have logging logic interspersed through otherwise decoupled business logic when woven. As even in aspect-orientation's most famous example there are benefits to the introduction of aspects woven within their targets, a study of the broader utility of the aspect-oriented metaprogramming approach should be conducted.

- Traditional aspects cannot make decisions based on reflection on specific properties of the code they are being applied to, such as calculations made, data accessed, computational complexity, and so on. There are many scenarios where one can anticipate this reflective behaviour to be useful. For example, compilation of Python code for efficient performance on a GPU (redirected from the CPU), as in cite the thesis here of Jeremy's student who worked in Python. He was very nice — can't remember his name for the life of me. , seems to decouple from the rewritten logic nicely in concept, but relies on an examination of iteration logic and specifics of the code being recompiled. Aspects with metaprogramming directly support reflection as access to the target's AST is trivial to achieve. There are many possible applications for this technology, and likely in a diverse set of domains; there is therefore research to be done to demonstrate the utility of this new approach.

- Would be nice to have a third reason as to why PyDySoFu introduces new research possibilities.

8.2 Future Work pertaining to RPGLite

RPGLite's dataset was analysed for the purposes it was collected for in this thesis: to aid in the realistic simulation of a well-controlled socio-technical system. However, many analyses are yet to be explored:

- Why were games abandoned? Are there patterns that can be identified which lead players to abandon games?
- Players likely formed cliques, where they would play against people they knew (perhaps in person) rather than relying on RPGLite's matchmaking features to find new opponents. The existence of cliques of players may have implications for the playstyles of players, how they learned "better" strategies over time, and the players' general dedication to playing RPGLite (and therefore producing a greater wealth of data for analysis and dissemination to the community)
- RPGLite's dataset contains information about players' interactions with the application itself; as the game made available some features typical of modern games (leaderboards, matchmaking, achievements, graphical customisation), an analysis of the features most commonly used can shed light on the more effective aspects of modern game design in both the general playerbase and more dedicated players[1]
- RPGLite's playerbase was recruited informally and there is scope for a larger and longer-term data collection effort to be made. A re-release of the application in major mobile app stores with a concerted effort to release new seasons of the game and maintain player interest for an extended period of time — perhaps with additional features, such as in-game chat, favourites lists –of previous opponents, or match replay and analysis (with suggestions for improved play backed by the formal methods inherant in RPGLite's design) would enable a richer analysis, and broader utility to the games research community.

While investigations into these questions warrant further study, they remain outwith the scope of this thesis, which focuses on simulation technologies more than it does game design. There are many opportunities available for the game design research community to investigate. Publications in the

[1] For example, are some features heavily used, but only by a dedicated subset? Do all players use other features a moderate amount, showing mild but general appeal?

field from co-creators of RPGLite reflect further on the design and future improvements of the game; see cite William's PhD here .

8.3 Future Work pertaining to Aspect-Oriented Simulation

Aspect-orientation's goal of separation of concerns, and the possibility of using its trait of obliviousness to augment naive models in ways the original creator did not anticipate, presents research opportunities that are also outwith the scope of this thesis.

- Myriad models exist which have been provided accurate results in past research, but could not account for unforseen modern situations. For example, models of world health over time could not account for the Covid19 pandemic, and models of the world economy could not incorporate real-world data from the recession caused by responses to the pandemic, or the 2008 financial crisis. The World3 model is an example of one which has provided accurate predictions for decades, but could not account for incidents in modern times when constructed. Models such as these work from prior data which requires some adjustment as simulated time progresses to account for events of a large enough scale to disrupt their simulated system (here, global population, industry, food, resources, and pollution). An alternative to adjusting the models directly — adding cases at the relevant points in time to introduce "blips" in simulated data — is to construct aspects which represent global events such as pandemics, economic crises, and others such as war or famine. These can be modelled on real-world data, which we have shown in this research to produce realistic simulations. A proof-of-concept of the approach as applied to pre-existing models would start this work, and the augmentation of existing models to improve their accuracy can follow.
- Relatedly, aspects can represent anticipated future states so as to model their potential impact without modifying a known-good model of the world today. Future health and economic crises can be constructed as prospective changes to a model in an aspect, and applied to investigate the possible effects. A potential benefit of this approach as opposed to the simple modification of an existing model would be that many potential crises can be applied in any combination. For example, 10 aspects representing unpredictable future events yield 1024 possible combinations:

there are 2^{10} possible combinations of these aspects being applied or omitted from an execution of a simulation. Work to develop aspect-oriented models of speculative futures therefore gives an exponential number of predicted futures, which one could analyse to predict possible future trends. With a successful proof-of-concept of the augmentation of existing models to represent past events, this further step could anticipate future events and take advantage of aspect orientation's unique properties as a tool for simulation and modelling.

→ Combinations of traits in human modelling where a player might not fit to one specific trait well, but fits to a combination applied weakly.

8.4 Testing Frameworks to detect Unrealistic Behavioural Variances

Given we don't know the impact of variances exactly, something like sciunits could give us "bounds" on realism in our model, i.e. the sciunit should encode limits on what the real-world system does, and let us know whether those limits are broken when applying variances (or combinations of them)

8.5 Standards for Model Features

How can we make a more "principled" approach to adopting features for modelling? PDSF's approach to adding things to a model is powerful and flexible. A technique that would work for broad applications could allow for a standard in the RSE community for simulations, at least for sociotechnical simulation. What are the best practices around the modification of scientific models in this way, that could encourage collaboration and the sharing of models across groups?

8.6 Optimisation of multiple models

An explanation of the future work in section 6.4.1, where we suggest that it'd be interesting future work for somebody to anneal to multiple models of aspectually applied behavioural variance.

The work to do on this point is relatively trivial — just a grid search on many dimensions really — but we've not done it and it'd risk detracting from our goal anyway, which is to show that we can optimise a model (so we should keep things from being unnecessarily complicated!) so worth leaving for an honours / masters dissertation.

8.7 Discussion

This section is not intentionally left blank.

Bibliography

[+06] hridesh rajan hridesh et al. *nu: towards an aspectoriented invocation mechanism*. Tech. rep. technical report 414, iowa state university, department of computer science, 2006.

[ABZ08] AU Aksu, Faruk 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.

[AGL98] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. “Mining process models from workflow logs”. In: *Advances in Database Technology—EDBT’98: 6th International Conference on Extending Database Technology Valencia, Spain, March 23–27, 1998 Proceedings 6*. Springer. 1998, pp. 467–483.

[BH02] Jason Baker and Wilson C. Hsieh. “Maya: Multiple-Dispatch Syntax Extension in Java”. In: *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*. PLDI ’02. Berlin, Germany: Association for Computing Machinery, 2002, pp. 270–281. ISBN: 1581134630. DOI: 10.1145/512529.512562. URL: https://doi.org/10.1145/512529.512562.

[Bir99] Colin PD Birch. “A new generalized logistic sigmoid growth equation compared with the Richards growth equation”. In: *Annals of botany* 83.6 (1999), pp. 713–723.

[Bui14] JCAM Buijs. “Flexible evolutionary algorithms for mining structured process models”. In: *Technische Universiteit Eindhoven* 57 (2014).

[Cap+09] Claudia Cappelli et al. “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. URL: https://doi.org/10.1145%2F1509825.1509828.

[CBB13] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. “Toward an aspect-oriented simulation”. In: *International Journal of New Computer Architectures and their Applications (IJNCAA)* 3.1 (2013), pp. 1–10.

[CBB14] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. “Practical benefits of aspect-oriented programming paradigm in discrete event simulation”. In: *Modelling and Simulation in Engineering* 2014 (2014).

[CBB19] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. “Using aop in discrete event simulation: A case study with japrosim”. In: *International Journal of Applied Mathematics, Computational Science and Systems Engineering* 1 (2019).

[Cla73] R Lawrence Clark. “LINGUISTIC CONTRIBUTION TO GOTO-LESS PROGRAMMING”. In: *Datamation* 19.12 (1973), pp. 62–63.

[CM06] Anis Charfi and Mira Mezini. “Aspect-Oriented Workflow Languages”. In: *OTM Conferences*. 2006.

[CM07] Anis Charfi and Mira Mezini. “Ao4bpel: An aspect-oriented extension to bpel”. In: *World wide web* 10.3 (2007), pp. 309–344.

[CMM10] Anis Charfi, Heiko Müller, and Mira Mezini. “Aspect-Oriented Business Process Modeling with AO4BPMN”. In: *Modelling Foundations and Applications*. Ed. by Thomas Kühne et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 48–61. ISBN: 978-3-642-13595-8.

[con16] RafeKettler & github contributors. *magicmethods (A guide to Python’s magic methods)*. https://github.com/RafeKettler/magicmethods/tree/65cc4a7bf4e72ba18df1ad17a377c 2011 — 2016.

[con21] Clarete & github contributors. *ForbiddenFruit*. https://web.archive.org/web/20210515092416/https://github.com/clarete/forbiddenfruit. 2021.

[CS04] Ruzanna Chitchyan and Ian Sommerville. “Comparing dynamic AO systems”. In: 2004.

[CSS04] Constantinos Constantinides, Therapon Skotiniotis, and Maximilian Stoerzer. “AOP Considered Harmful”. In: *In Proceedings of European Interactive Workshop on Aspects in Software (EIWAS)*. 2004.

[DBM13] Jan De Houwer, Dermot Barnes-Holmes, and Agnes Moors. “What is learning? On the nature and merits of a functional definition of learning”. In: *Psychonomic bulletin & review* 20 (2013), pp. 631–642.

[DD82] Edsger W Dijkstra and Edsger W Dijkstra. “On the role of scientific thought”. In: *Selected writings on computing: a personal perspective* (1982), pp. 60–66.

[dev21] SimPy developers. *SimPy 4.0.1 documentation*. https://simpy.readthedocs.io/en/4.0.1/. 2021.

[DGM08] Adrian Dozsa, Tudor Girba, and Radu Marinescu. “How lisp systems look different”. In: *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE. 2008, pp. 223–232.

[Dij68] Edsger W Dijkstra. “Letters to the editor: go to statement considered harmful”. In: *Communications of the ACM* 11.3 (1968), pp. 147–148.

[Don20] Boudewijn van Dongen. “BPI Challenges: 10 years of real-life datasets”. In: *IEEE Task Force on Process Mining Newsletter #2* (May 14, 2020). URL: https://www.tf-pm.org/newsletter/newsletter-stream-2-05-2020/bpi-challenges-10-years-of-real-life-datasets (visited on 05/14/2020).

[DOR95] DOV DORI. “Object-process Analysis: Maintaining the Balance Between System Structure and Behaviour”. In: *Journal of Logic and Computation* 5.2 (Apr. 1995), pp. 227–249. 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. URL: https://doi.org/10.1093/logcom/5.2.227.

[DR10] Robert Dyer and Hridesh Rajan. “Supporting dynamic aspect-oriented features”. In: *ACM Transactions on Software Engineering and Methodology* 20.2 (Aug. 2010), pp. 1–34. DOI: 10.1145/1824760.1824764. URL: https://doi.org/10.1145%2F1824760.1824764.

[Esp04] Miklós Espák. “Aspect-Oriented Programming On Lisp”. In: *International Conference on Applied Informatics*. 2004.

[FBS17] Daniel Friesel, Markus Buschhoff, and Olaf Spinczyk. "Annotations in Operating Systems with Custom AspectC++ Attributes". In: *Proceedings of the 9th Workshop on Programming Languages and Operating Systems*. 2017, pp. 36–42.

[FFN00] Robert E Filman, Daniel P Friedman, and Peter Norvig. "Aspect-oriented programming is quantification and obliviousness". In: (2000).

[FT+84] James France, John HM Thornley, et al. *Mathematical models in agriculture*. Butterworths, 1984.

[GK99] László Gulyás and Tamás Kozsik. "The use of aspect-oriented programming in scientific simulations". In: *Proceedings of Sixth Fenno-Ugric Symposium on Software Technology, Estonia*. 1999.

[Gom15] B. Gompertz. *On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contingencies.* Jan. 1815. DOI: 10.1098/rspl.1815.0271. URL: https://doi.org/10.1098/rspl.1815.0271.

[GS04] Wasif Gilani and Olaf Spinczyk. "A family of aspect dynamic weavers". In: *Dynamic Aspects Workshop (DAW04)*. 2004.

[Hil+00] Erik Hilsdale et al. "AspectJ: The Language and Support Tools". In: *Addendum to the 2000 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum)*. OOPSLA '00. Minneapolis, Minnesota, USA: Association for Computing Machinery, 2000, p. 163. ISBN: 1581133073. DOI: 10.1145/367845.368070. URL: https://doi.org/10.1145/367845.368070.

[Ion+09] Tudor B Ionescu et al. "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, pp. 21–30.

[JS07] Chris W Johnson and Christine 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.

[JWO12] Amin Jalali, Petia Wohed, and Chun Ouyang. "Aspect oriented business process modelling with precedence". In: *International Workshop on Business Process Modeling Notation*. Springer. 2012, pp. 23–37.

[KDB91] Gregor Kiczales, Jim Des Rivieres, and Daniel G Bobrow. *The art of the metaobject protocol*. MIT press, 1991.

[Kel08] Stephen Kell. "A survey of practical software adaptation techniques." In: *J. Univers. Comput. Sci.* 14.13 (2008), pp. 2110–2157.

[KH98] Ralph Keller and Urs Hölzle. "Binary component adaptation". In: *European Conference on Object-Oriented Programming*. Springer. 1998, pp. 307–329.

[Kic+01] G. Kiczales et al. "An Overview of AspectJ". In: *ECOOP*. 2001.

[Kic+97] Gregor Kiczales et al. "Aspect-oriented programming". In: *European conference on object-oriented programming*. Springer. 1997, pp. 220–242.

[KM20] William Kavanagh and Alice Miller. "Gameplay Analysis of Multiplayer Games with Verified Action-Costs". In: *The Computer Games Journal* 10.1-4 (Dec. 2020), pp. 89–110. DOI: 10.1007/s40869-020-00121-5. URL: https://doi.org/10.1007%2Fs40869-020-00121-5.

[KM21] William Kavanagh and Alice Miller. "Gameplay analysis of multiplayer games with verified action-costs". In: *The Computer Games Journal* 10 (2021), pp. 89–110.

[KWM20] William Kavanagh, Tom Wallis, and Alice Miller. "RPGLite player data and lookup tables". In: (2020).

[Lac97] Sheldon J Lachman. "Learning is a process: Toward an improved definition of learning". In: *The Journal of psychology* 131.5 (1997), pp. 477–480.

[LC07] Gary T Leavens and Curtis Clifton. "Multiple concerns in aspect-oriented language design: a language engineering approach to balancing benefits, with examples". In: *Proceedings of the 5th workshop on Software engineering properties of languages and aspect technologies*. 2007, 6–es.

[Li10] Chen Li. "Mining process model variants: Challenges, techniques, examples". PhD thesis. University of Twente, The Netherlands, 2010.

[Mac+11] Idarlan Machado et al. "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. URL: https://doi.org/10.1145%2F1960502.1960508.

[MAG12] Giselle Machado, Vander Alves, and Rohit Gheyi. "Formal Specification and Verification of Well-formedness in Business Process Product Lines". In: *6th Latin American Workshop no Aspect-Oriented Software Development: Advanced Modularization Techniques, LAWASP*. Vol. 12. 2012.

[Mar94] Robert Martin. "OO design quality metrics". In: *An analysis of dependencies* 12.1 (1994), pp. 151–170.

[Mat08] Norm Matloff. "Introduction to discrete-event simulation and the simpy language". In: *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2.2009 (2008), pp. 1–33.

[Mit+17] Alexey A. Mitsyuk et al. "Generating event logs for high-level process models". In: *Simulation Modelling Practice and Theory* 74 (May 2017), pp. 1–16. DOI: 10.1016/j.simpat.2017.01.003. URL: https://doi.org/10.1016%2Fj.simpat.2017.01.003.

[MK03] Hidehiko Masuhara and Gregor Kiczales. "A modeling framework for aspect-oriented mechanisms". In: *Proceeding ECOOP'03*. 2003.

[Nie77] Jurg Nievergelt. "Information Content of Chess Positions". In: *SIGART Bull.* 62 (Apr. 1977), pp. 13–15. ISSN: 0163-5719. DOI: 10.1145/1045398.1045400. URL: https://doi.org/10.1145/1045398.1045400.

[Obj11] Object Management Group (OMG). *Business Process Model and Notation, Version 2.0*. OMG Document Number formal/2011-01-03 (https://www.omg.org/spec/BPMN/2.0). 2011.

[Obj15] Object Management Group (OMG). *Unified Modelling Langauge, Version 2.5*. OMG Document Number formal/2015-03-01 (http://www.omg.org/spec/UML/2.5). 2015.

[OLe20] Aran O'Leary. "Simulating Human Behaviour in a Micro-epidemic using an Agent-based Model with Fuzzing Aspects". In: *Unpublished masters thesis* (2020).

[PAG03] Andrei Popovici, Gustavo Alonso, and Thomas 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.

[Par72] D. L. Parnas. "On the criteria to be used in decomposing systems into modules". In: *Communications of the ACM* 15.12 (Dec. 1972), pp. 1053–1058. DOI: 10.1145/361598.361623. URL: https://doi.org/10.1145%2F361598.361623.

[PGA02] Andrei Popovici, Thomas Gross, and Gustavo Alonso. "Dynamic Weaving for Aspect-Oriented Programming". In: *Proceedings of the 1st International Conference on Aspect-Oriented Software Development*. AOSD '02. Enschede, The Netherlands: Association for Computing Machinery, 2002, pp. 141–147. ISBN: 158113469X. DOI: 10.1145/508386.508404. URL: https://doi.org/10.1145/508386.508404.

[Pou+15] Asef Pourmasoumi et al. "On Business Process Variants Generation." In: *CAiSE Forum*. 2015, pp. 179–188.

[Prz10] Adam Przybylek. "What is Wrong with AOP?" In: *ICSOFT (2)*. Citeseer. 2010, pp. 125–130.

[RIC59] F. J. RICHARDS. "A Flexible Growth Function for Empirical Use". In: *Journal of Experimental Botany* 10.2 (June 1959), pp. 290–301. ISSN: 0022-0957. DOI: 10.1093/jxb/10.2.290. eprint: https://academic.oup.com/jxb/article-pdf/10/2/290/1411755/10-2-290.pdf. URL: https://doi.org/10.1093/jxb/10.2.290.

[Ros97] Jarrett Rosenberg. "Some misconceptions about lines of code". In: *Proceedings fourth international software metrics symposium*. IEEE. 1997, pp. 137–142.

[SA13] Thomas Stocker and Rafael 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.

[SA14] Thomas Stocker and Rafael Accorsi. "SecSy: A Security-oriented Tool for Synthesizing Process Event Logs." In: *BPM (Demos)*. 2014, p. 71.

[Seo11] Simon Seow. "OBASHI White Paper". In: *APMG-International, High Wycombe, UK* (2011).

[Sil+20] Hercules Sant Ana da Silva Jose et al. "Implementation of Aspect-oriented Business Process Models with Web Services." In: *Bus. Inf. Syst. Eng.* 62.6 (2020), pp. 561–584.

[SL07] Olaf Spinczyk and Daniel Lohmann. "The design and implementation of AspectC++". In: *Knowledge-Based Systems* 20.7 (2007), pp. 636–651.

[SM14] Ivan Shugurov and Alexey A Mitsyuk. "Generation of a set of event logs with noise". In: *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*. 8. □□□□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□□ □□□□□ □□□□□□□□□ □□□□□□□□□□□ … 2014.

[Ste06] Friedrich 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. URL: https://doi.org/10.1145%2F1167473.1167514.

[Sul01] Gregory T Sullivan. "Aspect-oriented programming using reflection and metaobject protocols". In: *Communications of the ACM* 44.10 (2001), pp. 95–97.

[SW16] Tim Storer and William Wallis. *FUZZI-MOSS*. https://web.archive.org/web/20201018121800/https://github.com/twsswt/fuzzi-moss. 2016.

[Tha+17] Dai Hai Ton That et al. "Sciunits: Reusable Research Objects". In: *2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, Oct. 2017. DOI: 10.1109/escience.2017.51. URL: https://doi.org/10.1109%2Fescience.2017.51.

[Van+05] Boudewijn F Van Dongen et al. "The ProM framework: A new era in process mining tool support". In: *Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings 26*. Springer. 2005, pp. 444–454.

[VD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[Ver45] PF Verhulst. "La loi d'accroissement de la population". In: *Nouveaux Memories de l'Académie Royale des Sciences et Belles-Lettres de Bruxelles* 18 (1845), pp. 14–54.

[Von38] Ludwig Von Bertalanffy. "A quantitative theory of organic growth (inquiries on growth laws. II)". In: *Human biology* 10.2 (1938), pp. 181–213.

[VVK09] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. "The refined process structure tree". In: *Data & Knowledge Engineering* 68.9 (2009), pp. 793–818.

[VW04] Wil MP Van der Aalst and Anton JMM Weijters. "Process mining: a research agenda". In: *Computers in industry* 53.3 (2004), pp. 231–244.

[Wal+21] William Wallis et al. "Designing a mobile game to generate player data - lessons learned THIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXMETHIS IS A DUPLICATE FIXMETHIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXME THIS IS A DUPLICATE FIXME Duplicate citation to clean up ". In: *CoRR* abs/2101.07144 (2021). arXiv: 2101.07144. URL: https://arxiv.org/abs/2101.07144.

[WC10] Paul Wallis and Fergus Cloughley. *The OBASHI Methodology*. The Stationery Office, 2010.

[WJ97] AR Werker and KW Jaggard. "Modelling asymmetrical growth curves that rise and then fall: Applications to foliage dynamics of sugar beet (Beta vulgarisL.)" In: *Annals of Botany* 79.6 (1997), pp. 657–665.

[WS18a] Tom Wallis and Tim 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.

[WS18b] Tom Wallis and Tim Storer. "Process Fuzzing as an Approach to Genetic Programming". In: *Proceedings of the SICSA Workshop on Reasoning, Learning and Explainability*. Ed. by Kyle Martin, Nirmalie Wiratunga, and Leslie S. Smith. Vol. 2151. CEUR Workshop Proceedings. Aberdeen, UK: CEUR-WS.org, June 2018. URL: http://ceur-ws.org/Vol-2151/Paper_S3.pdf.

[YC79] Edward Yourdon and Larry L Constantine. "Structured design. Fundamentals of a discipline of computer program and systems design". In: *Englewood Cliffs: Yourdon Press* (1979).