Daniel Moyo, Abdallah K. Ally, Alan Brennan, Paul Norman, Robin C. Purshouse and Mark Strong (2015)

Agile Development of an Attitude-Behaviour Driven Simulation of Alcohol Consumption Dynamics

Journal of Artificial Societies and Social Simulation 18 (3) 10
<http://jasss.soc.surrey.ac.uk/18/3/10.html>

Received: 15-Apr-2014 Accepted: 15-Apr-2015 Published: 30-Jun-2015

Abstract

Whilst there have been several advocates for the application of software engineering (SE) methodologies in the development of agent-based models and simulations in the social sciences, the uptake of these techniques in the research community has been limited – or if authors are using such techniques, their use is underreported. Software engineering provides structured processes and techniques for designing, documenting, implementing and testing computer software. Software processes have many variations, each with their own unique advantages and disadvantages depending on the constraints (such as: human resources, time, finance, quality) facing a project team. This paper sets out the methods of Scrum agile software development, and discusses the experience of using Scrum to organise workflow and guide the development of an agent-based model of alcohol consumption. By employing Scrum in conjunction with another software engineering method, the Unified Modelling Language, this paper represents a case study in SE methods applied to a real world research problem.

Keywords:
Agile, Agent-Based, Alcohol, Attitudes, Microsimulation, Modelling

Introduction

1.1 Agent-based modelling and simulation (ABMS) is a method of computational research with an established history in fields as diverse as economics (Cristelli et al. 2011), biology (An et al. 2009), and social science (Gilbert 2007) where it is often labelled agent-based social simulation (ABSS). Computer programs that perform ABSS are examples of simulation software.

1.2 In order to be used with confidence as a scientific instrument, it is important that any software be engineered with quality and rigor; however, not all developers of simulation software have the experience or knowledge of best practices for software development, although some introductory guides exist outside of the software development literature that are specifically related to creating software for scientific research (Baxter et al. 2006; Wilson et al. 2014).

1.3 The discipline relating to the practice of principled software creation is known as Software Engineering (SE). Despite the wealth of available literature on SE, the plethora of SE techniques have yet to receive widespread acceptance and adoption within the context of ABMS and ABSS (Klügl 2003). Principled SE offers many benefits to ABSS developers. Its overriding purpose is to facilitate the development of software, ensuring that software can be used, updated and extended upon by individuals other than those involved in its development (Sommerville 2009). The term computer program is often used synonymously with the word software; however, software is much more, and is intended to represent all of the relevant data and documentation associated with the operation of a given computer program.

1.4 In this paper we will present a case study in the use of a software process known as *agile development* for the purpose of developing a computational model of alcohol consumption. We will first introduce software engineering and explain why academia has unique challenges in relation to the production and maintenance of software. We will then describe some of the techniques that agile methods provide that may be of use to those developing software in an academic environment, with a particular focus on a variant called *Scrum* (Schwaber & Sutherland 2013).

1.5 After motivating the use of complex systems methods for the development of a model of alcohol consumption, we will describe our modelling methodology and explain how Scrum was used as a means of driving our project through fostering stakeholder engagement and team communication, organising work flow, and increasing developer productivity. We will then reflect on the use of Scrum for our project. The use of established software engineering practices such as the *Unified Modelling Language* (OMG 2011) to document and design simulation software will also be discussed in the context of our alcohol modelling case study. Finally, we will briefly present our vision for the research theme, building upon our current model to incorporate social dynamics into future software iterations, whilst adhering to the agile method.

Software Development, Processes, and Applicability in an Academic Setting

Software development in academia

2.1 In the academic world software is developed for a wide variety of purposes, and by individuals or groups with a wide range of previous expertise in the production of software. Researchers often develop code as individuals, or in small teams, and perhaps with their own programming styles. These developers may or may not be aware of the various methods and processes that have been established for best practice in creating, documenting and maintaining software. Documentation is quite often a necessity to enable future staff employed on the project to become familiar with a pre-existing piece of software, though planning and documenting software involves a significant investment of time and resources.

2.2 Several issues face those who develop software for academic purposes. The unstable nature of academic software projects, with the uncertainty of funding timescales and staffing arrangements (e.g. via fixed term contracts), places a demanding time constraint on the development process, and creates problems for the future maintenance and extension of existing projects and code-bases (Pitt-Francis et al. 2008).

2.3 Developers make decisions on how to create software conditional on a range of factors, including prior experience and current resources (time, human resources, finances). Quite often developers, regardless of experience, are often tempted to code without a plan regarding how they will manage their resources; writing a program, its functions and algorithms on the fly until an apparently working program emerges. Such an approach has many critical disadvantages. Firstly, without specifying what the software is required to do, or a list of *features* (distinguishing software characteristics or functionality), it is not possible to measure objectively whether or not the software fulfils its purpose. Secondly, without a description of how or why the software has been designed or constructed, future users or developers of the software may struggle to maintain, extend or test it. Thirdly, without any consideration for time-management, it becomes extremely difficult to deliver a quality software product with all of the desired functionality and within the duration of the project. SE approaches address these issues by forcing developers to either formally document the software requirements and design, or by encouraging teams to work closely with customers and stakeholders to iteratively develop software; with customer or user feedback guiding future software iterations. SE also often adopts principles from project management, providing methods for structuring the various aspects of the development process.

Software processes

2.4 A plethora of software processes exist that those developers with knowledge of SE might be aware of. Software processes are frameworks that contain a set of discrete activities that aid the software development process (Sommerville 2009), and one of the more traditional software process frameworks is the *Waterfall* model. The Waterfall model is a set of activities that are tackled in a linear fashion. Firstly a specification of *requirements* is created, which are then translated into a set of *software requirements* and a *system/architectural software design* constructed to fulfil those requirements. The design is subsequently programmed into computer code in the *implementation* phase. Finally, some form of *verification* is performed to ensure the implementation satisfies the design and requirements. Verification activities can be either static (e.g. code checks, formal verification of algorithms) or dynamic checks (e.g. actively testing the functionality of the program and/or its constituent components) (Dasso & Funes 2007).

2.5 It can be argued that the basic waterfall model described above, alongside similar variants, is particularly inflexible as a paradigm for developing software when project variables are subject to a degree of change; as each activity in the waterfall model must be complete before moving onto the next. With any software project, there may be design errors, or a change in requirements, and with such a linear approach it can become time-consuming to reconstruct designs, and update all of the relevant project documents and outputs. The dynamic nature of academic research has additional concerns that make it incompatible in some respects with such a linear approach to development. New insights are often gleaned, or the research changes direction due to new data or hypotheses. It then becomes more difficult to introduce design changes with such a heavily front-loaded approach to requirements specification and design.

2.6 Several alternative software process models exist, including but not limited to: *rapid application development* (RAD), *incremental development* and *spiral*, each with their own advantages and disadvantages given the specific context in which they will be used (Sommerville 2009). An alternative set of principles that takes inspiration from some of these processes is a group of methods known as *agile* development methods, which contain several techniques that can address many of the issues facing those developing software in an academic setting.

Agile development methods

2.7 The over-riding principles of agile development are iteration and flexibility. Agile is a people-oriented approach that centres on prioritising the requirements dictated by the project's various stakeholders and agreed by the developers. Iterative in nature, agile requires a workforce that is dynamic to change, which can be introduced more rapidly with smaller development teams such as those in an academic setting. Agile may not be the best method as team sizes scale, and there are those that believe agile methods are not suited to larger development teams (Cohen et al. 2004).

2.8 The values of agile are documented in the *agile manifesto* (Manifesto for Agile Software Development 2001), and one of those values relates to software documentation: agile practitioners believe that *working software* takes priority over *comprehensive documentation*. Practitioners of agile often therefore adopt a *lean* approach to documentation, and rely on the knowledge of experienced team members to pass on information relating to software design and implementation to new developers (Pitt-Francis et al. 2008). Such a practice assumes that there will always be someone working on a project who has intimate knowledge of its details, which may not be the case with academic projects, which may suffer from periods of inactivity or changing staffing arrangements. Therefore, it is up to the developers to decide on the appropriate level of documentation required, without significantly impacting on the agility of the team.

2.9 There are several variations on the agile themed approach, each recommending its own set of activities to be performed. A recent systematic review describes the body of agile development literature, and the reader is directed there for a more comprehensive review of the various methods available (Dybå and Dingsøyr 2008). In realising the simulation software presented later in this paper, we took inspiration from the *Scrum* development method (Schwaber & Sutherland 2013). Scrum inherits many themes from *iterative and incremental development* (Larman & Basili 2003), and the process actively assumes that the development cycle will be turbulent and changing, and provides methods that are adaptive to those changing project requirements (Schwaber 1997).

Scrum

2.10 Scrum comprises three core components: (1) the *Product Backlog*; (2) the *Sprint Backlog*; and (3) the *Working Software Increment* (Figure 1). Firstly a Product Backlog is created, comprising all of the known features required in the final software product. Since Scrum is iterative, the Product Backlog is not a fixed set of features or requirements. The backlog can be adapted depending on the changing requirements of the relevant stakeholders, or due to practical or technical issues, for example bug fixing (e.g. arithmetic, logic, or syntax errors in the code) of the current Working Software Increment, or implementing previously unplanned yet newly essential or desirable features. Secondly, a Sprint Backlog is created, providing a prioritised list of the desired software work items and features for the upcoming Sprint. A Sprint is simply a dedicated cycle of work with a planned duration of time to implement items from the Sprint Backlog; typically a Sprint lasts between 1–4 weeks in duration. During each Sprint, the project team should ideally have a Daily Scrum, a short meeting to plan work activities for the day. The Working Software Increment is the product of each Sprint and should have useful functionality that can be discussed by the project team in order to inform future iterations of the software.

2.11 Alongside the various Scrum components, there are three roles in any Scrum project. The *Product Owner* represents the customer or any stakeholders with an interest in the final product. The *Development Team* is an umbrella term for all individuals involved in the software development of the product. Finally, the *Scrum Master* ensures that the Scrum process is adhered to, and prioritises features to be included in the next Sprint cycle.

2.12 A target is set for each Sprint, called the *Sprint Goal.* Each Sprint Goal is agreed upon during a *Sprint Planning* meeting, which is an extended meeting to establish what will be implemented in the next Sprint, and how the team will accomplish that implementation (). At the end of a Sprint, a *Sprint Review* is undertaken to evaluate the current Working Software Increment, detailing to the stakeholders what has been accomplished, and providing an opportunity for the wider team to be updated on the current status of the project. The wider team represents those individuals who may contribute to certain aspects of the project, perhaps having been called on for their expertise, but are not involved in the day-to-day running of the project or have ultimate responsibility for delivery of project aims and objectives.

2.13 A Scrum Planning Board is created to keep track of project tasks at various stages of development. The planning board can be implemented physically, with a whiteboard and sticky notes, providing an excellent focal point for a team to engage in their Scrum meetings. Alternatively, there are a wide variety of online tools and graphical packages that can be used to implement a Scrum board collaboratively or otherwise. Either the physical or graphical planning board method provides a visual means of keeping track of the project, organised by Product Backlog, Sprint Backlog, in progress (features of current Sprint) and completed

features. For a team or lone developer, having a planning board allows active visualisation of the flow of activities day-to-day, providing a positive sense of accomplishment as tasks move from the backlog to completion. Should items linger in the backlog for too long, rather than become disheartened, developers may take the time to re-evaluate the status of those items according to whatever prioritization method the project team agrees on, thus maintaining feature flow and productivity.

2.14 Scrum as a method has been adopted by academics within several different contexts. Specific to developing software for research, the Cancer, Heart and Soft Tissue Environment (CHASTE) was developed to provide a simulation framework suitable for computational models within biology (Mirams et al. 2013). The CHASTE project team recognised the pitfalls of developing software within academia and chose to adhere to a test-driven agile approach, focusing on activities such as pair programming (two developers coding together at one computer) which they argued helps team cohesion and promotes collective ownership (Pitt-Francis et al. 2009). In another study, a novel Scrum approach to managing a research group comprising a cohort of postgraduate research students was developed (Hicks & Foster 2010a). The approach, known as Scrum for Research (SCORE), encourages regular (three-times-per-week) status updates where the whole group is in attendance, and recommends more in-depth *on-demand* meetings for any immediate challenges that arise for group members (Hicks & Foster 2010b). The authors report that SCORE increased group productivity, allowed ideas to be shared and elaborated upon rapidly, and helped students cope with stresses relating to feeling a lack of progress with their individual works. Within the field of pedagogy, Scrum has also been used to structure a games development course according to a set of design (pre-production) and Sprint (game production) activities, and to promote teamwork through regular team interaction (Schild et al 2010). The flexible, iterative Scrum approach suited the evolving nature of games development, allowing prototype development to feed new requirements for the Product Backlog, whilst organising the project workload into Sprints designed to fit with the schedule of the curriculum. The diverse range of applicable areas in which agile can be applied to academia makes it a promising method for software development purposes and more generally, project management.

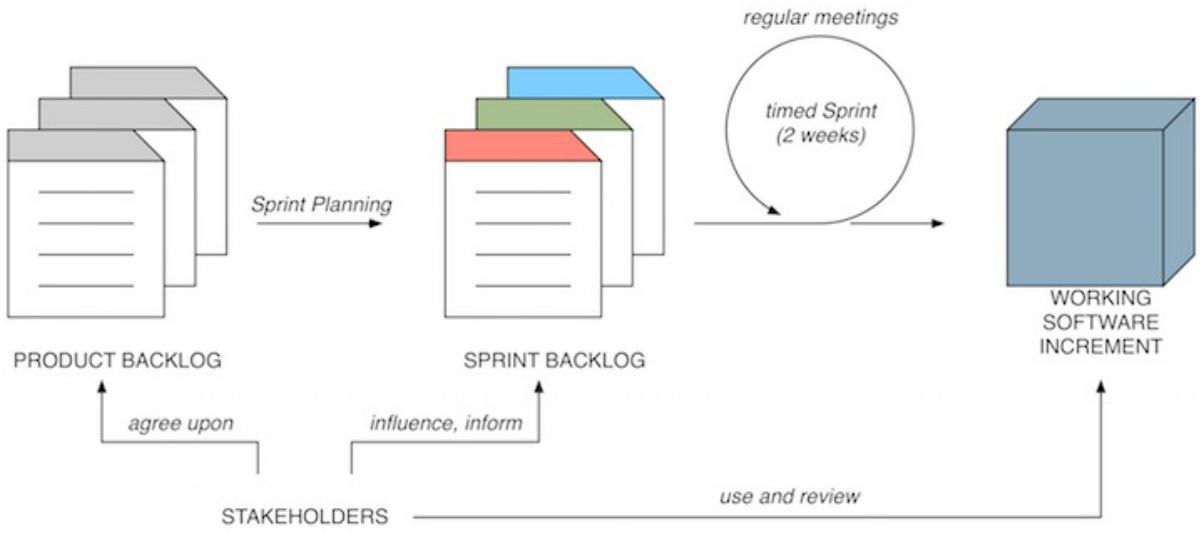

Figure 1. The Scrum process. Stakeholders (including developers) agree upon the required features and functionality of the final software product and these comprise the Product Backlog. The stakeholders have a Sprint Planning meeting to determine those work items from the Product Backlog that will form the Sprint Backlog. Work items are then implemented from the Sprint Backlog by the developers in time-boxed Sprints. Stakeholders review the Working Software Increment during a Sprint Review meeting, and this informs the development of future software iterations.

Unified Modelling Language

2.15 As previously discussed, the traditional *lean* approach to documentation with agile assumes project members will remain to impart their knowledge to future developers, which whilst ideal is not always possible in academia. In these instances, documentation is necessary to ensure the long-term viability of the project independent of specific developers.

2.16 One method of describing and documenting software is to use the Unified Modelling Language (UML) (OMG 2011). UML, traditionally used in the field of software engineering, has 14 separate diagrams (as of UML v2.4) that can be used to visually model a system, its objects and their relationships. The utility of each and every UML diagram is a matter of debate, and their utility is ultimately down to the belief of the modeller in each diagram's ability as a communicative medium. There are several proponents of UML who describe both benefits and pitfalls of UML notation for the design of ABMs of a variety of systems (Bauer & Odell 2005; Bersini 2006, 2012; Read et al. 2009; Alden et al. 2012).

Unit testing

2.17 Verification, the act of ensuring that software adheres to its specification, is an important activity in the development of any software – this is particularly important when the proposed software is to be used for the purposes of scientific experimentation. Unit testing is one method of software verification, and is central to many software development methodologies including the agile approach known as Extreme Programming (XP). Unit tests can be written to determine whether the various methods and functions that comprise the software operate as expected and required. Unit tests also may provide valuable insight to developers new to an existing software project. Tests are generally designed to expose the software to both valid and invalid input data, and act in part as a form of documentation for the software, or at least for the constituent components for which unit tests have been written.

Revision control

2.18 Another common practice, essential for the preservation and maintenance of code during the software project life cycle, is revision control (RC), allowing the storage, management and documentation of changes to code (Osborne et al. 2014). The key benefit of RC is that it provides a stable means of modifying a project's code base without overwriting previous versions of that code. This is achieved by creating new versions of a set of initial files, called a *revision.* Each individual file can be reverted to, or merged with, any previous revision. Should the developer have a body of code associated with a working piece of software, to avoid breaking the existing code, a branch of duplicate code can be create from the existing code base. Updates are then applied to the new branch and these changes can be later merged into the original branch, subsequent to the relevant testing and verification, or be kept as stand-alone software versions called *forks*. Some RC tools provide the ability to chart the evolution of a software product visually, providing a valuable resource to current and future

developers, detailing all of the software revisions and forks.

2.19 Online source code repositories are available for RC, and several of these allow users to open up their code to the public, lending transparency to the scientific process by allowing researchers to more easily share their code and provide insight into its development (Prlic & Procter 2012), and also facilitating reproducibility of simulation research (Sandve et al. 2013). Commonly used repositories include *GitHub* (GitHub, Inc. www.github.com) and *Bitbucket* (Atlassian www.bitbucket.org). Repository providers have their own individual pricing strategies and offer a range of additional features, though most have a free to use option available.

Complex Systems Modelling of Alcohol Consumption Dynamics: A Case Study in Scrum

3.1 The research detailed in this paper represents a work package within an academic research project: *Complex Systems Modelling of Alcohol Consumption Dynamics in the British Population (CSMACD)*, funded by the UK Economic and Social Research Council. The aim of the work package was to identify empirically validated, causally-driven, micro-simulations that bridge the gap between micro-social (individual-level) and macro-social (population-level) knowledge of the dynamics of drinking

3.2 Whilst computational models are relatively well established within the field of alcohol policy appraisal, contemporary models tend to assume that consumption change is purely a function of ageing (Chisholm 2004; Hollingworth et al. 2006; Purshouse et al. 2014). However recent age-period-cohort analyses have indicated strong birth cohort and period effects: i.e. today's 45-54 year olds do not exhibit consumption patterns similar to 45–54 year olds in previous decades (Meng et al. 2014). Furthermore, when considering behaviour change arising from an intervention, these contemporary models rely on neoclassical economics – specifically, they assume that individual drinkers behave rationally, have access to perfect information, and are able to maximize their own utility subject to a budget constraint. The validity of these assumptions as a credible generating mechanism for drinking change is questionable (Hedström 2005) and ignore wider theories on personal and social factors that drive alcohol as a complex system (Holder 1998; Room et al. 2009). In summary, no quantitative model has been established that convincingly explains how individual-level drinking can manifest the observed population-level trends in alcohol consumption patterns.

3.3 We hypothesise that ABMS can allow us to provide new insight into historical patterns of alcohol consumption, together with potential predictive capability of future trends, by allowing us to investigate how individual behaviours and social interactions might influence the system. Such an approach would harness the benefits of the agent-based approach within a social science context (Bonabeau 2002; Gilbert 2008) via the emergence of complex macro-level patterns (i.e. alcohol consumption trends) arising from the drinking behaviour of interacting individuals at the micro-level (Bonabeau 2002; Gilbert 2004; Hedström 2005).

3.4 ABM represents a powerful analytical tool to understand both causality and emergence within the social sciences (Hedström 2005) and agent-based social network models have been used in a number of alcohol related studies (Fowler 2009; Ormerod & Wiltshire 2009; Giabbanelli & Crutzen 2013). Our initial goal was to ensure that our model was data-driven and constructed using evidence-based assumptions regarding individuals and the psychological drivers of alcohol consumption, before moving towards a model including social interaction. The reason for this is that in order for any complex systems model to be accepted within the alcohol field, and to gain acceptance for alcohol policy appraisal, it needs to have strong data-driven and evidence-based foundations (Katikireddi et al. 2014). Once such foundations are in place, we can then incorporate elements of theory and hypothesis-driven experimentation regarding social connectedness, social establishments and frameworks, to determine how interactions influence drinking behaviour, as it is argued that a combination of both social psychology and social interaction theory can provide a greater level of explanatory power with regards to sociological research (Hedström 2005).

Am agile approach to complex systems modelling of alcohol consumption dynamics

3.5 An agile approach was chosen for the software development stages of the project for two reasons. Firstly, the project was designed to use secondary data (principally from the UK Data Archive) that was not collected for the specific research questions to be addressed by the study. Consequently, parameterisation issues were anticipated when attempting to extract model inputs for theory-led (and typically data hungry) ABMs (Boero & Squazzoni 2005). These issues induce attempts to negotiate or balance modelling requirements relating to theory on the one hand and empirical validation on the other: in other words, requirements will change. Secondly, the project had a short (12 month) time constraint, requiring an approach that allowed for rapid prototyping during the development stages (8 months). We also wanted to develop a software framework that is extensible, enabling the project to be maintained in future research efforts.

The project team

3.6 The project team initially consisted of four members: the principal investigator and lead modeller (Robin), the modeller and lead developer (Daniel) and two domain experts – Paul, our social psychologist, and Alan, a specialist in modelling and decision-making in the health domain. These four members of our interdisciplinary project team are the key stakeholders in the project. Alan, Daniel, Robin and Paul decided the features for our Product Backlog. Our primary developer Daniel acted as the Scrum master and prioritised features for the next Sprint. Two additional researchers (Abdallah and Mark) were called upon for their modelling expertise since the inception of the project, and were considered part of the wider project team, and did not play an active role in the Scrum process, though were kept informed as to the status of the project as it progressed.

Prioritising software features

3.7 To prioritise all possible model features, we adopted the Must Should Could Would (MoSCoW) method (Clegg & Barker 1994). MoSCoW is designed to ensure that both developers and project stakeholders have agreed upon the importance of the features of a piece of software. MoSCoW priority in relation to CSMACD model development is as follows:

3.8 *Must have* (MH) features are those that are essential for our alcohol model to function, and to be able to simulate an output measure related to consumption. These features include those relating to: loading and parameterisation from individual-level respondent data, algorithms to impute data from additional data sets, functions to update the states in the model, and methods required to extract useful data from the simulation.

3.9 *Should have* (SH) features are those that are not critical to the operation of our alcohol model; however, they should eventually be implemented. These can include, for example, additional output metrics (see section 3.4); functionality to perform additional analyses on the model, such as analysing simulator robustness to parameter perturbation; or implementing optimisation algorithms to explore and optimise system parameterisations (Purshouse et al. 2014). Refactoring may also feature as an SH activity. Refactoring is the act of restructuring computer code without changing the overall function of that code, and is a valuable and required activity for any developer. Small scale refactoring, perhaps to improve the readability of a method or algorithm, is required to facilitate re-usability and extensibility of existing code. More large scale, and therefore time consuming, refactoring activities may be required, though their importance may be lower than more pressing features, and would thus fall into a lower category of priority.

3.10 *Could have* (CH) features are desirable but can be omitted if project resources or time are scarce. As highlighted as a SH feature, refactoring could also be a marked as a CH feature if the activity requires substantial effort and resource to accomplish. Note that since refactoring is not the act of changing the function of the code, it is up to the developer to evaluate the priority of such an activity. For example, if an algorithm had numerous lines of duplicate code, a developer might wish to refactor to remove the superfluous code from the implementation, thus improving maintainability of the code and perhaps indirectly the efficiency

of their code through resource recovery. Depending on how detrimental the original implementation is to the operation of the program, such a refactoring activity might be more suited to a CH activity than a SH activity, especially if development human resources are scarce.

3.11 *Would have* (WH) features are those that might be useful for future increments of the software, but that are not currently scheduled to be taken forward (Brennan 2009). They may be similar to CH features, but that are identified as outwith the scope of the development process due to project constraints.

3.12 Figure 2 provides a basic example of a Scrum planning board incorporating the MoSCoW method, with some sample features from our project. It was implemented physically with a whiteboard and sticky notes.

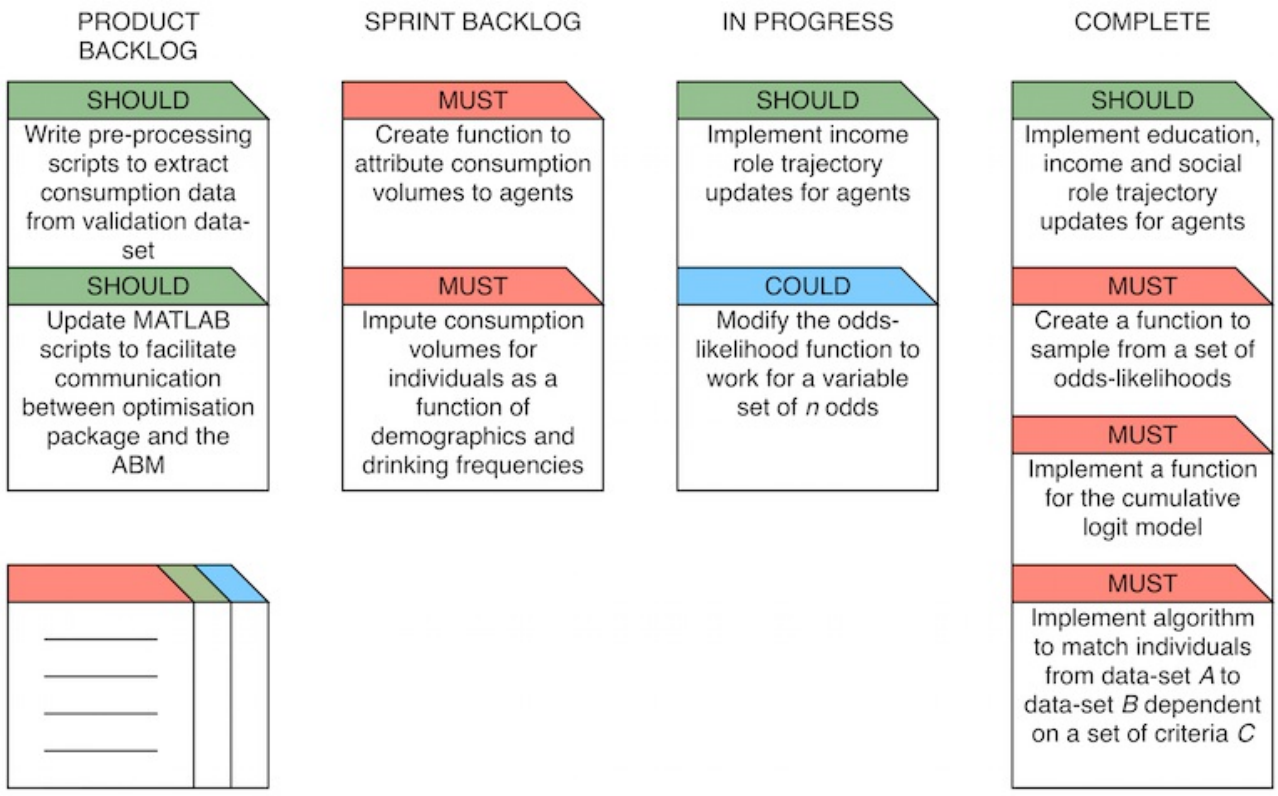

Figure 2. A sample Scrum planning board with MoSCoW prioritised work items. Intuitively, one would assume that all *must have* items are implemented first; however, due to the iterative nature of agile, and ever changing product features, not all must have features or work items are identified early, and can often be added later in the development process with different priorities.

Overview of the Sprints

3.13 Prior to our first Sprint, we performed a scoping study, which involved a review of the literature relating to established behaviour models from psychology, and also an extensive search through the data literature to find data that could be used to parameterise the first iteration of our model. This scoping study and the construction of our domain model took place over the first 3 months of the project, we report on the development Sprints for the remaining 9 months of the project for which we produced a working model after each iteration.

3.14 We chose to implement an attitude-behaviour model that takes inspiration from the Theory of Planned Behaviour (TPB) (Azjen 1991). TPB presumes that behaviour is determined by an intention to perform that behaviour, in our case engaging in a drinking occasion or drinking to intoxication. Intention itself is determined by three core components: *attitudes* towards that behaviour based on the individual's (positive or negative) evaluations of the likely outcomes of the behaviour, *subjective norms* relating to how an individual's peer group (parents, partners, friends) feels about the individual performing the behaviour, and lastly *perceived behavioural control* which relates to the perceptions the individual has regarding their ability to perform the behaviour, as well as constraints relating to performing that behaviour, such as income (which impacts on affordability of alcohol).

3.15 In creating any computational model, parameterisation from data sources is vital from both a practical and a validation perspective. In the context of our research, parameterisation takes the form of using individual-level respondent data to operationalise a model of TPB. Upon analysis of the available resources in the UK Data Archive, it became evident that we could not ideally parameterise the TPB. The only population-wide source of alcohol related attitudinal variables is the *Offending Crime and Justice Survey 2003* (Home Office 2008). The decade between 2000 and 2010 has seen several interesting population-level phenomena emerge (e.g. a rise and subsequent fall in consumption for the general population) and so this period of time was deemed a useful window over which to simulate.

3.16 Analysis of the OCJS data set revealed variables detailing the self-reported frequency of drinking (fdrink) and frequency of drinking to intoxication (fdrunk) status. Intoxication is an important area of study as it is associated with a variety of acute alcohol related harms including: road traffic accidents (Ridolfo & Stevenson 2001), both unintentional (English et al. 1995) and intentional (English et al. 1995; Single et al. 1996) injuries, and also absenteeism from work (Roche et al. 2008). However fdrunk was deemed of less priority in the Sprint as historical fdrunk data for validation purposes is not available.

3.17 For the platform implementation of our model, subsequent to an initial scoping study and domain analysis, we opted to code the model without reliance on existing agent-based modelling platforms or libraries. Whilst the use of existing libraries might offer efficiency benefits in terms of code reuse, we preferred the additional flexibility offered by a fully bespoke solution and had the programming skills in the team to make such an approach feasible.

3.18 In all, we undertook six development sprints during the research effort. The balance of time was spent writing manuscripts (including for this Special Issue of the *Journal of Artificial Societies and Social Simulation*), conference visits and separate macro-level dynamic modelling for both the current project and an additional collaborative project with research partners. We summarize the six sprints below.

First iteration: an object-oriented agent-based microsimulation of drinking frequency

Sprint goal

3.19 The OCJS was chosen as a data set to operationalise TPB in our model, with consumption represented in the dataset by an individual's fdrink. The goal for this Sprint was to develop an agent-based model recreating fdrink trends in England between 2003 and 2009.

Sprint planning and the Sprint

3.20 In our project, each Sprint was time boxed with a duration of one month. For this Sprint, agents within our ABM represent individual respondents from the OCJS, and encode traditional demographic variables (age, gender, education), as well as TPB-related agent parameters: alcohol related attitudes (individual drinks to: feel relaxed, forget problems, feel friendly/outgoing, get drunk); a proxy for norms (count of the types of groups an individual drinks with); and proxies for perceived control (specific social roles held by the individual, the number of unique places the individual normally drinks in, and income). We define social roles as parenthood (dependent child in household), partnership (cohabitation) and paid labour (holding a salaried income) – denoted PPP.

3.21 During the Scrum Planning meeting, the team took advice from a statistical expert (Abdallah) who identified that a cumulative logit model (CLM) (Agresti 2013) would be suitable for modelling changes in the categorical fdrink variable. Based on this discussion, several features required to implement a CLM were added to our Product and Sprint Backlogs.

3.22 To introduce dynamics to the CLM, input parameters must vary. Aging is an obvious change that would occur during the simulation; however, aging could only go so far in explaining the dynamics of agent drinking states. We therefore anticipated the need for alternative sources of dynamics, such as changes in: education, income, and social role statuses. These variables can be measured empirically, though changes in them are driven by wider social phenomena, therefore predicting variable trajectories is not a trivial problem. As a result, the stakeholders (Alan, Daniel, Paul and Robin) agreed on features designed to incorporate exogenous inputs into our model using empirically observed demographic and social role trajectories. The trajectory feature was deemed a SH feature, as the model could function without such exogenous inputs; it would just lack face validity with regards to individual dynamics.

3.23 Trajectories were acquired from a secondary data set, the *British Household Panel Survey (BHPS)* (ISER 2010). The BHPS is a longitudinal study with available data having been acquired between 1991 and 2009. These data provide trajectories for education (NFQ levels) (Ofqual 2011), income, as well as the three social roles. Upon initialisation, the simulation matches agents (OCJS individuals) to trajectories extracted from the BHPS based on an exact set of criteria (age-group, gender, education, income and PPP status). Individual matches are then sampled from at run-time and the trajectories extracted from those matches provide dynamics for the underlying CLM.

3.24 The CLM is fitted in R 3.0.2 and the intercepts and coefficients are used as parameters for the agent-based simulation. The ABM is implemented in Python v2.7.5. For the case study outlined in this paper we utilise Bitbucket as the company's free pricing package also offers a private repository option accessible to 5 individuals, which was useful when our software was in the early stages of development and not ready to be shared. All code related to this project can be cloned from our Bitbucket repository (https://pyabm@bitbucket.org/pyabm/pyabm.git).

Sprint review

3.25 The first iteration was completed with no major problems; however, the matching algorithm had a performance issue. The algorithm was a crude implementation that, for every individual in the OCJS data set, then searched every individual in the BHPS data set to find a match, storing matches for each individual. Such an approach is computationally expensive and involves N*M comparisons (length of BHPS * length of OCJS), and led to a significant increase in simulation initialisation time. As a result, we decided to add a CH feature to the next Sprint, which was to tune the matching algorithm to improve simulator performance.

Second iteration: parameter estimation of the TPB model

3.26 A hypothesis we were interested in exploring was whether there existed multiple models that could explain historical drinking, but which could render very different future trajectories of consumption – based on ideas in Byrne (1998). To this end, we conceived a second sprint to seek out such models using computational intelligence methods.

Sprint goal

3.27 The goal was to develop an evolutionary optimizer, incorporating niching (Goldberg & Richardson 1987), capable of identifying parameterisations of the CLM that describe observed fdrink dynamics between 2003–2009.

Sprint planning and the Sprint

3.28 The research team – specifically Robin – has expertize in evolutionary optimization and has an existing toolbox developed for the Matlab environment. Given the time constraints of the sprint, we decided to re-use functionality from the toolbox. Since the ABM was written in python, an early task in this Sprint for Daniel was to develop a Matlab wrapper for the ABM. In parallel, Robin developed the Matlab scripts and, with the wrapper in place, performed the analysis.

Sprint review

3.29 The optimizer was able to find a family of models that could recreate historical drinking trends. The one-step-ahead predictions of each model were combined

to provide an ensemble forecast for fdrink in 2010. The method and results have been published in Purshouse et al. (2014).

Third iteration: modelling drinking to intoxication

3.30 Having established a baseline model that was capable of simulating patterns in fdrink over time, we were now interested in introducing additional fdrunk functionality, and in improving the performance of the matching algorithm.

Sprint goal

3.31 The Sprint Goal was to introduce fdrunk functionality into the existing model, and implement various code optimisations to improve the performance of the simulation.

Sprint planning and the Sprint

3.32 The matching algorithm code optimisation was upgraded from a CH to a MH feature, as the performance degradation of the original algorithm increased the time required to test and run the model. A caching approach to the matching criteria was used to reduce the complexity of the algorithm.

3.33 Implementing the fdrunk functionality involved fitting a CLM to predict probabilities of fdrunk states conditional on fdrink. Erroneous data was removed, for example, respondents who report an fdrink of *once a month* yet an fdrunk of *most days*. Additional intercept and coefficient parameters were added to the simulation, along with implementations of functions to calculate fdrunk probabilities and to sample from them.

Sprint review

3.34 During the previous sprint, the team undertook a planned research visit to the Centre for Addiction and Mental Health (CAMH), Toronto, to engage alcohol epidemiological experts in the project. A new MH requirement emerged from the visit: ensuring that the model – amongst its drinking patterns outputs – included a measure of *alcohol exposure* (in grams of ethanol per day). Such a measure would enable the model to be linked to existing epidemiological models that forecast volumes of alcohol-related harm for policymakers. Whilst this had been a CH requirement originally, the team at CAMH felt that the new modelling was sufficiently advanced and useful for work on the epidemiological interface to be accelerated. By raising the requirement to MH, the project would be well-positioned for future use as a tool for policy appraisal relating to reducing alcohol-related harms.

Fourth iteration: introducing measures of consumption

3.35 We had initially considered incorporating a consumption metric into our model during the original scoping study; however, we could not find a data set containing both the alcohol related attitudinal data and consumption data. As a result we focussed mainly on patterns of drinking and using the OCJS as our primary data set. After feedback from CAMH we decided to revisit the issue of consumption.

Sprint goal

3.36 The fourth iteration of the model aimed to incorporate an alcohol exposure measure into the ABM. Average consumption data is present in a variety of studies, for example the *Health Survey for England (HSE)* (NatCen and UCL 2011) and the *General Lifestyle Survey (GLF)*, though not present in our primary data set, the OCJS. A third data set was therefore required to allow us to impute plausible levels of exposure for individuals. We decided to change our primary data set to the GLF for two reasons. Firstly, the survey sample size is greater than the OCJS after ensuring data completeness, giving a more representative population sample. Secondly, the GLF contains mean weekly alcohol units as a consumption metric, allowing us to estimate a model that we could use to predict year-on-year consumption.

Sprint planning and the Sprint

3.37 Drawing on further statistical expertise (Mark), we constructed a Box-Cox regression (BCR) model that describes exposure for GLF individuals based on their age, age group, gender, education, income, parenthood, partnership, paid labour, fdrink, and various interaction terms between these variables as covariates. The BCR model was fitted in R 3.0.2. Change in exposure is then driven by changes in demographics and the predicted year-on-year drinking frequency. As a SH feature in the Sprint Backlog, we aimed to implement the BCR model in python so that consumption levels were calculated within the ABM rather than performed afterwards.

3.38 With the GLF now acting as the primary data set, TPB-related agent parameters were imputed from the OCJS using a similar matching approach to that previously described. Life-course trajectories were imputed from the BHPS as previously described.

3.39 We previously described a requirement for our project that we document the function of our alcohol model in order to ensure the long-term viability of our software independent of developer. Whilst we do not retrospectively create extensive software requirements documents, we do utilise modelling tools to describe our simulation software, specifically UML. The class diagram in Figure 3 depicts a simple relationship between the two core system classes (excluding a utility class for data input/output etc.). Figure 3 also contains two activity diagrams that concisely depict the sequence of activities or events that drive the internal initialisation and operation of both the simulation and its multiple agent objects.

Sprint review

3.40 One of our intentions was to implement the BCR model in python, though because of the in-Sprint improvements, including stepwise model selection by AIC and correlated errors, and the collaborative effort with Mark who is familiar with R and not python, we decided to downgrade the priority of re-implementing BCR into the python model to a CH feature.

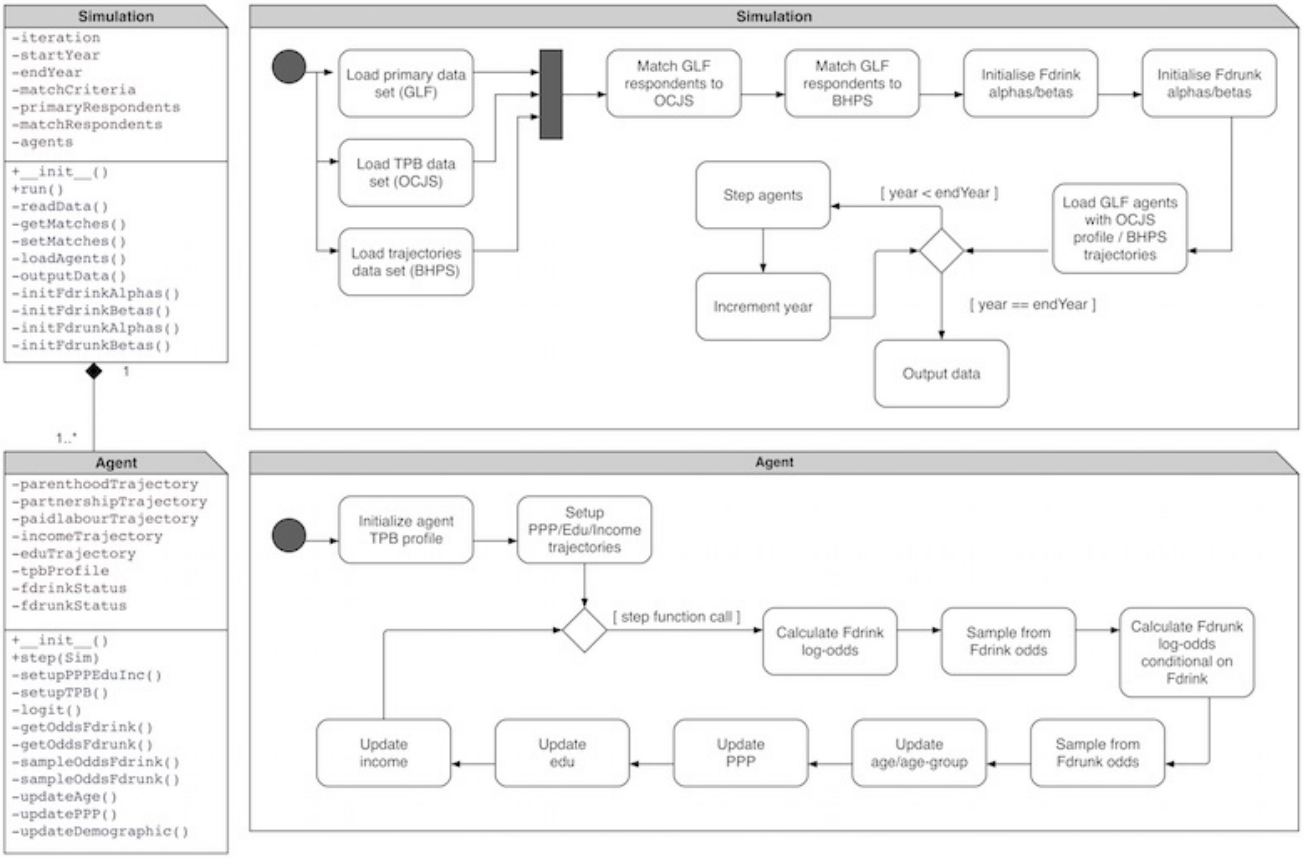

Figure 3. Class and activity diagrams representing simulation and agent objects for model iteration three.

Fifth iteration: simulating attitude change

3.41 We were unable to identify a UK data source which tracks attitude variables related to alcohol consumption over time. This is unfortunate since, for alcohol, attitudes have been shown to be strongly correlated with behaviours in TPB studies (Cooke et al. 2014). We decided to revisit the issue of attitude change and determine any alternative means of modelling attitudes.

Sprint goal

3.42 The goal of this sprint was to adopt two methods for dynamic individual-level attitude change. Firstly, the aim was to adopt a scenario analysis approach using simple assumptions about attitude responses of individuals either *increasing, decreasing,* or *varying* over time. Secondly, using the cross-sectional data from the OCJS in 2003, we constructed Markov models to identify transition probabilities for attitude responses conditional on respondent age.

Sprint planning and the Sprint

3.43 The scenario analysis approach to attitude change was conceptually the most straightforward of our two approaches to implement, though is evidently not data-driven in any respect. The motivation for exploring this method of attitude change was to determine if such blanket assumptions regarding attitude change could more accurately explain observed fdrink and consumption patterns over the baseline, and to also provide a comparison against the second approach using transition probabilities.

3.44 The transition probabilities approach was a more complex modelling endeavour, and involved careful consideration of transition rules (structure of allowed transition matrices) for the four attitudes included in our model, each with four point attitude responses ranging from 0 (disagree strongly) to 3 (agree strongly). A parameter estimation technique was employed, similar to the method described in Purshouse et al. (2014), to search over a range of transition probabilities that could explain the OCJS 2003 observed attitude responses grouped by age.

Sprint review

3.45 We were able to find transition probabilities to explain the OCJS 2003 data, however, including those into our model was insufficient for the model to accurately explain dynamics in fdrink and consumption across the simulated period. The same was true for two of the three scenarios in our scenario analysis. The model which included decreasing attitude responses (increasingly negative responses regarding alcohol for all attitude variables) proved the most able at capturing trends in fdrink, particularly the decrease in *most days* drinking seen over the past decade (the heaviest drinking category). The cross-sectional attitude model is the more data-driven of the approaches, but the failure of such a model to replicate observed data reveals one of two things; that either the cross-sectional data is not representative of attitude change, or that the model is inherently missing certain factors to allow it to more accurately predict history.

Sixth iteration: accounting for stochastic variance

3.46 Previous research during our second Sprint had demonstrated a use for evolutionary parameter estimation for finding different candidate operationalisations of our microsimulation model that help to explain observed historical drinking frequency data, however, our findings did not account for stochastic uncertainty in the underlying microsimulation, and also did not evaluate consumption. Aleatory analysis (Alden et al. 2012) later determined a minimum of 250 replicate microsimulation runs in order to mitigate the effects of stochastic uncertainty in the microsimulation, and after performing 250 simulation runs we can generate a stable median prediction of frequency of drinking. Therefore, for a given set of initial parameters, our consumption estimator needed to be able to process data from 250 simulation runs to find the most representative model outputs.

Sprint goal

3.47 This Sprint aimed to update our consumption estimator to process the additional volume of simulation data produced since our aleatory analysis stipulates a minimum of 250 simulation runs for any experiment. Both the ABM and the estimator in essence form a hybrid model which must be capable of being invoked using our evolutionary optimizer.

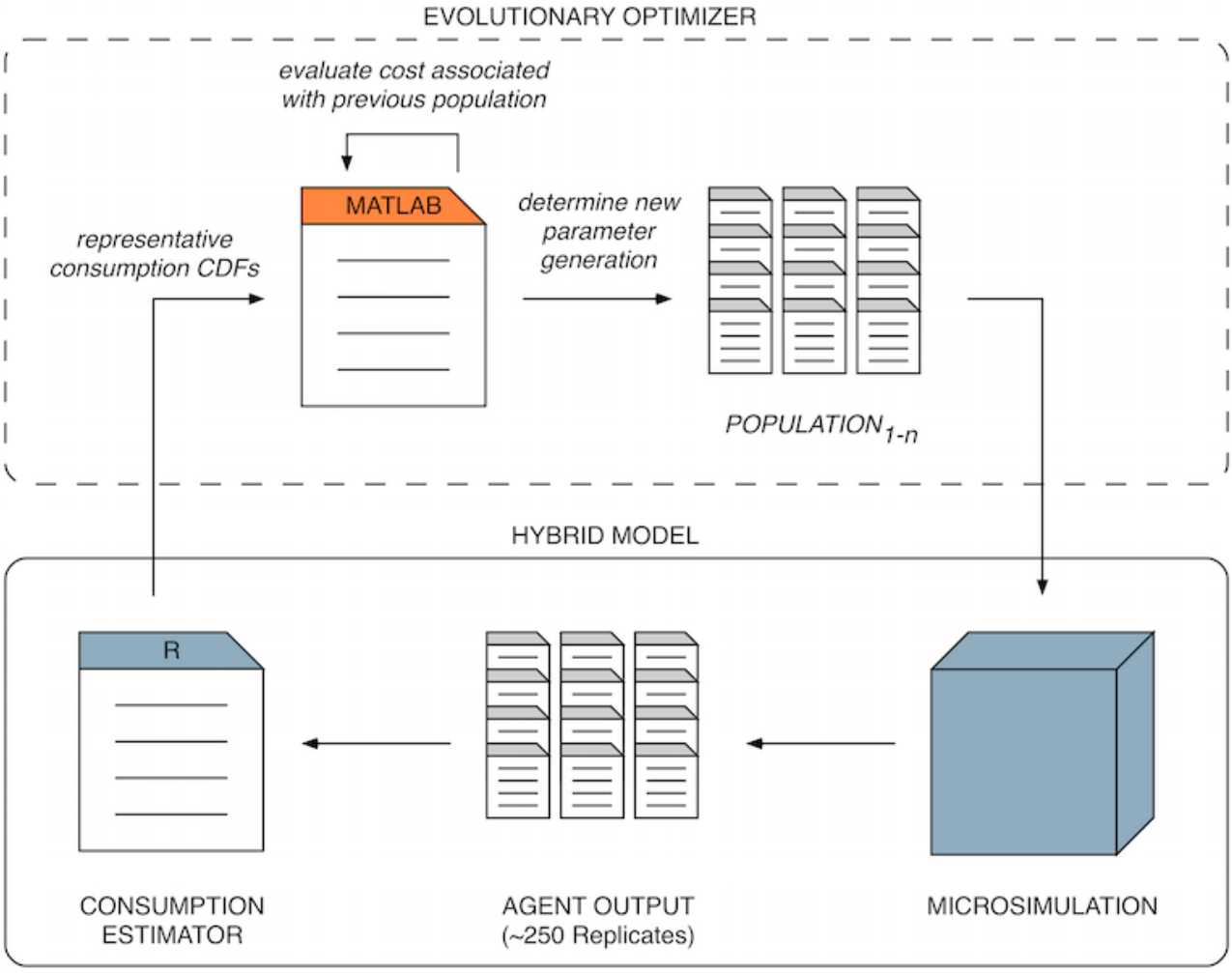

Figure 4. Parameter estimation and evaluation of alcohol consumption distributions using a hybrid model approach.

Sprint planning and the Sprint

3.48 This Sprint, focused on the hybrid model component of our proposed workflow, was split into three key phases. The first body of features were related to the creation and execution of compute cluster scripts to allow multiple agent-based microsimulation runs to be run in parallel using grid computing facilities. The second phase was to modify our consumption estimator script to evaluate all of the resulting agent output, compute consumption distributions for each replicate run, and output the most representative consumption distribution. The third phase is to integrate the hybrid model with and updated version of the evolutionary optimizer created during our second Sprint.

Sprint review and proposed sixth iteration

3.49 Whilst the Sprint was completed, a review of the fifth model iteration has highlighted a technical challenge that requires addressing in a future Sprint. Our initial exploration into the use of parameter estimation to determine candidate model parameters that best explain historical trends in fdrink was successful (Purshouse et al. 2014), and we had discussions around building upon that approach to explain historical trends in exposure in addition to fdrink. However, with the added execution time for the model to account for stochastic uncertainty, and the additional time required by our estimator (which now includes stepwise model selection to determine the best model fit for the data for each replicate simulation run), the execution time of our model during parameter estimation will increase by several orders of magnitude. The current optimization approach uses an evolutionary algorithm that is not tailored for expensive cost function evaluations, so a further sprint will be required to design an algorithm that can cope with a more limited number of calls to the ABM (e.g. through use of surrogate models). Figure 4 illustrates the proposed future workflow for our model, and will form the basis for the next Sprint.

Future Sprints

3.50 Agent interactions will be a key focus for future sprints. The OCJS data set that is used to parameterise some components of our individual agents contains information relating to the preferred drinking locations of individuals, and with whom they chose to drink. Such information will prove essential to maintaining a data-driven approach to our modelling effort, whilst at the same time providing a route into exploring location and social connectedness, providing the capacity of the ABSS model to exhibit emergent behaviour. The object-oriented agent-based approach we have adopted means that such additional functionality can be implemented whilst maintaining our existing code base. The Simulation and Agent classes (Figure 3), containing all the relevant code required to parameterise the model, load and schedule agents, perform agent behaviour, and output simulation data, will be maintained. The additional spatio-temporal and interaction functionality will be implemented by introducing concepts such as an abstract representation of an *environment*, which might itself comprise of a number of *locations* that are attributable to agents; these model aspects will be conceptualised using UML.

Discussion

4.1 Our case study in using the Scrum process to develop an ABM of the dynamics of alcohol drinking patterns has proved instrumental in us maintaining an evidence-based approach to developing our model. The compartmentalisation of work into Sprints has focussed on data gathering and modelling of individual behaviour prior to any consideration of more traditional benefits of an agent-based modelling approach, such as modelling environment and interaction. Whilst our ultimate goal is to make use of the benefits that ABM can bring, Scrum has forced us to strictly adhere to the scope of our initial requirements for our model: we have developed microsimulations that validate against historical dynamics for frequency of drinking. Further research efforts will aim to extend this approach to social interaction and possibly co-evolution – thus completing an adaptive generative account (Hedström 2005; Room 2011) that is grounded in the evidence base. The latter is crucial for acceptance of new methods in the alcohol policy research field.

Evaluation of Scrum

4.2 Scientific, empirical evaluation of the effectiveness and utility of agile methods for software development is rare (Lindvall et al. 2002), and where studies do exist they are limited to industry or near-industry sectors. Salo and Abrahamsson (2004) reported one such empirical review with a case study of the XP agile methodology; demonstrating how qualitative and quantitative data can be generated at different phases of XP agile, assisting in the evaluation of the process. Other attempts to compare agile methods quantitatively are limited to *on paper* attributes of agile processes. An example is the evaluation framework described by Calo et al. (2010), which the authors used to compare Scrum and XP agile based on the emphasis each process places on attributes such as individuals, teams, the process, tools, customer negotiation and documentation. According to their framework, XP is a more favourable agile methodology than Scrum (Calo et al. 2010, Table 2.). The issue is that no evaluation framework is considered the exemplar, and there is no way of determining how applicable each agile process is given a particular customer, their requirements, and set of stakeholders. Whilst empirical analysis of agile is needed, the collection of the required

evaluation data (e.g. stakeholder diaries, documented code defects, process review documents) is outside the scope of the current project.

4.3 Without a formal evaluation of the Scrum process compared to alternative SE methods, we cannot say categorically that Scrum is the most applicable option for ABSS. In our opinion, however, Scrum has brought benefits that have outweighed its costs. For example, in a busy academic environment occasions do arise when it is difficult to have an impromptu gathering of our interdisciplinary project team in order to resolve development problems. If stakeholders are not immediately available to sign off on any general requirements or feature changes, development can be delayed. In these instances, our developer had to adjust his daily work schedule to implement lower priority features, including those that might not have been scheduled for the current Sprint cycle, or improving existing code through refactoring. In such instances, Revision Control (RC) played an integral part in maintaining distinctly separate versions of the code, ensuring that non-functional changes (refactored code), and features implemented yet not stakeholder-agreed, remained distinct from the main Working Software Increment. At the next available stakeholder meeting, additions could then be signed off retrospectively; and the code merged into the Working Software Increment. Alternatively, should stakeholders wish not to include the already implemented functionality, the work item may remain in the Product Backlog ready for the relevant Sprint; with full or partial code instantly accessible through the RC repository. Another issue with Scrum is that it can be argued that there is a time overhead associated with the process (Scrum Planning/Scrum Review meetings, organising the planning board etc.); however, we have found that this time is more than recouped through the organisational efficiencies of using the process.

4.4 Predominantly, agile methods such as Scrum are more suited to small teams of developers, thus it may seem as if such methods are impractical for a one-two person development team such as ours; however, this is not the case as we argue below.

4.5 The principles that Scrum advocates, for example, prioritisation of tasks, visualising and managing progress, and incremental development, are ultimately beneficial as much to individual productivity as they are to the team's productivity. There have been occasions when Robin has also engaged as a developer; for example, in the second Sprint; however, this involved close collaboration with Daniel, as the optimiser had to directly interface with the ABM.

4.6 Adhering to Scrum forces our team to communicate regularly, and to facilitate this we have thrice weekly or more short (15 minute) status meetings between our developer and one or more stakeholders (predominantly Daniel and Robin), with more detailed fortnightly meetings with the members of the wider project team (Abdallah, Alan, Mark, and Paul). These meetings provide an opportunity for rapid feedback relating to the items in our Product Backlog, and allow our developer to carefully assess the priorities for current and upcoming Sprints. Our developer also prioritises daily tasks each morning, during a one-person Scrum. The one-person Scrum provides time to set the days goals, regardless of how trivial, and helps maximise individual productivity.

4.7 Based on our experience of using Scrum, we have a produced a list of simple recommendations on how ABSS developers can operationalise Scrum within their project (Table 1). We believe that Scrum could be extended to the full scope of a research project, including the planning and management of non-development related academic activities, such as manuscript preparation.

Table 1: Practical recommendations for the adoption of Scrum in an academic research project

	Recommendations when operationalising Scrum for a research project
1	Assign Scrum roles to the project team and familiarise them with The Scrum Guide (Schwaber & Sutherland 2013) to ensure members are aware of their responsibilities. This may be achieved with a half-day workshop or a hands-on tutorial session.
2	Establish a regular meeting schedule between project stakeholders and the development team. The development team should also organise as close to daily meetings as is feasible – this frequency of meetings should be adhered to, even for what might appear to be relatively long research work packages (e.g. 12 months).
3	Have a Scrum Master who follows the process as closely as is possible for your circumstances, despite any short-term pressures that might arise during the project to relapse to an unstructured approach.
4	Establish a Revision Control workflow and set up a source code repository before any coding begins.
5	Prioritise work items and functionality according to a method such as MoSCoW.
6	Fiercely adhere to the scope of each Sprint. Functionality can be always implemented in later Sprints; iteration is the key.

Conclusion

5.1 This paper has described the adoption of an agile approach to a real-world academic research project: the development of an agent-based microsimulation of the dynamics of alcohol consumption. The motivation for agile was driven by several project constraints including, amongst others, a small development team, data issues and a strict 12 month project duration. Requiring a development methodology that would be resistant to potential changes and disruptions in the project, and with a view to creating extendible and reusable software, agile methods were an appealing choice for our situation.

5.2 Despite the wealth of available agile methods, none were an ideal fit for our project team; however, we chose to take inspiration from principles of Scrum. Scrum has much in common with change-driven project life cycles in project management (PMI 2013) which setting software aside, provides methods to promote stakeholder engagement and deliver projects on time given a set of resources and constraints. Given that we anticipated changing software requirements, the Scrum process allowed us to organise our project, and alongside the MoSCoW method helped us prioritise work items to ensure incremental development of an early working prototype, adding features and fixing code bugs in a principled manner.

5.3 Purists might argue that the methods adopted by our small one-two person development team might fall under the umbrella of other agile approaches such as *personal kanban* rather than Scrum. Personal kanban is another agile process by which to both visualise and prioritise a body of required work (Benson & Barry 2011) although it does not directly include guidelines for stakeholder involvement since it is a methodology for individuals. Whilst it is true we are not adhering to the letter of Scrum regarding roles, which is not possible given the size of our development team, the way we structure our work cycle, prioritise software features, and communicate as a team, is most definitely in line with the Scrum ethos. Ultimately, a team or individual should take inspiration from whatever agile methods suit them and the needs of the project, with a goal to delivering high quality scientific software through iterative and incremental change.

5.4 Agile methods generally promote a lean approach to software documentation; however, for our study we chose to use UML to describe the function of our model. For our relatively straightforward implementation the UML is more than sufficient to convey the inner workings of the software to future developers. As model complexity increases, this approach may need re-evaluating, as the time overhead involved with updating software documentation to account for an evolving project may not be feasible.

5.5 In conclusion, we argue that more ABSS researchers should adopt SE methods in future, choosing whichever specific practices and variants they feel are of best use to them. More case studies in agile software development will provide the ABSS and wider academic software development fields with further qualitative and possibly quantitative evidence on the utility of those agile methods.

Acknowledgements

This work was funded by the UK Economic and Social Research Council under grant number ES/K001760/1. We would also like to thank researchers from the Centre for Research in Social Simulation (CRESS) at the University of Surrey and Jürgen Rehm and his team at the Centre for Addiction and Mental Health (CAMH) at the University of Toronto for helpful discussions, as well as the JASSS reviewers for their useful comments. The BHPS, GLF, HSE, and OCJS are Crown Copyright. The original data creators, depositors, or copyright holders, the funders of the data collections (when different), and the UK Data Archive bear no responsibility for analyses or interpretation of the data described in this report.

References

AGRESTI, A. (2013). *Categorical Data Analysis*. Wiley, New Jersey, 3rd edition.

AJZEN, I. (1991). The theory of planned behavior. *Organizational behavior and human decision processes*, 50(2), 179–211. [doi:10.1016/0749-5978(91)90020-T]

ALDEN, K., Timmis, J., Andrews, P. S., Veiga-Fernandes, H., & Coles, M. C. (2012). Pairing experimentation and computational modeling to understand the role of tissue inducer cells in the development of lymphoid organs. *Frontiers in immunology*, 3. [doi:10.3389/fimmu.2012.00172]

AN, G., Mi, Q., Dutta-Moscato, J., & Vodovotz, Y. (2009). Agent-based models in translational systems biology. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 1(2), 159–171. [doi:10.1002/wsbm.45]

BAUER, B., & Odell, J. (2005). UML 2.0 and agents: how to build agent-based systems with the new UML standard. *Engineering applications of artificial intelligence*, 18(2), 141–157.

BAXTER, S. M., Day, S. W., Fetrow, J. S., & Reisinger, S. J. (2006). Scientific software development is not an oxymoron. *PLoS Computational Biology*, 2(9), e87. [doi:10.1371/journal.pcbi.0020087]

BENSON, J., & Barry, T. D. (2011). *Personal Kanban: Mapping Work, Navigating Life*. Modus Cooperandi Press.

BERSINI, H. (2006). 'Immune system modeling: The OO way.' In *Artificial Immune Systems* (pp. 150-163). Springer Berlin Heidelberg. [doi:10.1007/11823940_12]

BERSINI, H. (2012). UML for ABM. *Journal of Artificial Societies and Social Simulation*, 15(1) 9: http://jasss.soc.surrey.ac.uk/15/1/9.html

BOERO, R. & Squazzoni, F. (2005). Does Empirical Embeddedness Matter? Methodological Issues on Agent-Based Models for Analytical Social Science. *Journal of Artificial Societies and Social Simulation*, 8(4) 6: http://jasss.soc.surrey.ac.uk/8/4/6.html

BONABEAU, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 3), 7280–7287. [doi:10.1073/pnas.082080899]

BRENNAN, K. (Ed.). (2009). *A Guide to the Business Analysis Body of Knowledge*. IIBA.

BYRNE, D. S. (1998). *Complexity Theory and the Social Sciences: An Introduction*. Routledge, Abingdon.

CALO, K. M., Estevez, E., & Fillottrani, P. (2010). A Quantitative Framework for the Evaluation of Agile Methodologies. *Journal of Computer Science & Technology (JCS&T)*, 10(2).

CHISHOLM D, Rehm J, Van Ommeren M, Monteiro M. (2004) Reducing the global burden of hazardous alcohol use: A comparative cost-effectiveness analysis. *Journal of Studies on Alcohol*, 65(6):782–793. [doi:10.15288/jsa.2004.65.782]

CLEGG, D., & Barker, R. (1994). *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc.

COHEN, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in computers*, 62, 1–66. [doi:10.1016/S0065-2458(03)62001-2]

CRISTELLI, M., Pietronero, L., & Zaccaria, A. (2011). Critical overview of agent-based models for economics. *arXiv preprint* arXiv:1101.1847.

COOKE, R., Dahdah, M., Norman, P., & French, D. P. (2014). How well does the theory of planned behaviour predict alcohol consumption? A systematic review and meta-analysis. *Health psychology review*, 1–53. [doi:10.1080/17437199.2014.947547]

DASSO, A., & Funes, A. (Eds.). (2007). *Verification, validation and testing in software engineering*. IGI Global. [doi:10.4018/978-1-59140-851-2]

DYBÅ, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9), 833–859. [doi:10.1016/j.infsof.2008.01.006]

ENGLISH, D. R., Holman, C. D. J., Milne, E., Winter, M., Hulse, G. K. et al. (1995). The quantification of drug caused morbidity and mortality in Australia. Commonwealth Department of Human Services and Health, Canberra.

FOWLER, F. J. (2009). *Survey research methods* (4th ed.). Sage.

GIABBANELLI, P., & Crutzen, R. (2013). An Agent-Based Social Network Model of Binge Drinking Among Dutch Adults. *Journal of Artificial Societies & Social Simulation*, 16(2) 10 : http://jasss.soc.surrey.ac.uk/16/2/10.html

GILBERT, N. (2004). Agent-based social simulation: dealing with complexity. *The Complex Systems Network of Excellence*, 9(25), 1–14.

GILBERT, N. (2007). 'Computational social science: Agent-based social simulation.' *Agent-based Modelling and Simulation*. Bardwell, Oxford, pp. 115-134

GILBERT, N. (2008). *Agent-based models* (No. 153). Sage, London.

GOLDBERG, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49.

HEDSTRÖM, P. (2005). *Dissecting the social: On the principles of analytical sociology*. Cambridge: Cambridge University Press. [doi:10.1017/CBO9780511488801]

HICKS, M., & Foster, J. S. (2010a). Score: Agile research group management. *Communications of the ACM*, 53(10), 30–31.

HICKS, M., & Foster, J. S. (2010b). Adapting Scrum to Managing a Research Group. *Technical Report, Department of Computer Science, University of Maryland*.

HOLDER, H. D. (1998). Planning for alcohol-problem prevention through complex systems modeling: Results from SimCom. *Substance use & misuse*, 33(3), 669Communications of the ACM692. [doi:10.3109/10826089809115890]

HOLLINGWORTH, W., Ebel, B. E., McCarty, C. A., Garrison, M. M., Christakis, D. A. & Rivara, F. P. (2006). Prevention of deaths from harmful drinking in the United States: the potential effects of tax increases and advertising bans on young drinkers. *Journal of Studies on Alcohol*, 67(2): 300-308.

HOME OFFICE. Research, Development and Statistics. Directorate. Offending Surveys and Research, National Centre for Social Research and BMRB. Social Research. Offending, Crime and Justice Survey, 2003 [computer file]. 3rd Edition. Colchester, Essex: UK Data Archive [distributor], August 2008. SN: 5248.

ISER (Institute for Social and Economic Research, University of Essex) (2010). British Household Panel Survey: Waves 1-18, 1991-2009 [computer file]. 7th Edition. Colchester, Essex: UK Data Archive [distributor], July 2010. SN: 5151.

KATIKIREDDI SV, Hilton S, Bonell C, Bond L. (2014) Understanding the development of minimum unit pricing of alcohol in Scotland: A qualitative study of the policy process. *PLoS ONE*, 9(3):e91185. [doi:10.1371/journal.pone.0091185]

KLÜGL, F. (2013). "Engineering" Agent-Based Simulation Models?. In *Agent-Oriented Software Engineering XIII* (pp. 179-196). Springer Berlin Heidelberg. [doi:10.1007/978-3-642-39866-7_11]

LARMAN, C., & Basili, V. R. (2003). *Iterative and incremental development: A brief history.* IEEE Computer Society. [doi:10.1109/MC.2003.1204375]

LINDVALL, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., ... & Zelkowitz, M. (2002). 'Empirical findings in agile methods.' In Extreme Programming and Agile Methods—XP/Agile Universe 2002 (pp. 197–207). Springer Berlin Heidelberg. [doi:10.1007/3-540-45672-4_19]

MANIFESTO for Agile Software Development. (2001) http://www.agilemanifesto.org/. Archived at: http://www.webcitation.org/6Or4vOCBA

MAY N. R. (2011). Implementing eResearch Projects using Agile Development. In *Proceeding of eResearch Australasia*.

MENG, Y., Holmes, J., Hill-McManus, D., Brennan, A., & Meier, P. S. (2014). Trend analysis and modelling of gender-specific age, period and birth cohort effects on alcohol abstention and consumption level for drinkers in Great Britain using the General Lifestyle Survey 1984–2009. *Addiction*, 109(2): 205-215.

MIRAMS, G. R., Arthurs, C. J., Bernabeu, M. O., Bordas, R., Cooper, J., Corrias, A., ... & Gavaghan, D. J. (2013). Chaste: an open source C++ library for computational physiology and biology. *PLoS computational biology*, 9(3), e1002970. [doi:10.1371/journal.pcbi.1002970]

NATIONAL Centre for Social Research and University College London. Department of Epidemiology and Public Health. Health Survey for England, 2009 [computer file]. 2nd Edition. Colchester, Essex: UK Data Archive [distributor], July 2011. SN: 6732.

OFFICE of the Qualifications and Examinations Regulator (Ofqual). Qualifications can cross boundaries – a rough guide to comparing qualifications in the UK and Ireland. 3rd edition, 2011.

OMG. (2011). Object Management Group Unified Modeling Language^{TM} (OMG UML), Infrastructure V2.4.1.

ORMEROD P. & Wiltshire, G. (2009). 'Binge' drinking in the UK: a social network phenomenon. *Mind & Society*, 8(2): 135-152.

OSBORNE JM, Bernabeu MO, Bruna M, Calderhead B, Cooper J, et al. (2014). Ten Simple Rules for Effective Computational Research. *PLoS Comput Biol*, 10(3), e1003506. [doi:10.1371/journal.pcbi.1003506]

PITT-FRANCIS, J., Bernabeu, M. O., Cooper, J., Garny, A., Momtahan, L., Osborne, J., ... & Gavaghan, D. J. (2008). Chaste: using agile programming techniques to develop computational biology software. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1878), 3111–3136.

PITT-FRANCIS, J., Pathmanathan, P., Bernabeu, M. O., Bordas, R., Cooper, J., Fletcher, A. G., ... & Gavaghan, D. J. (2009). Chaste: a test-driven approach to software development for biological modelling. *Computer Physics Communications*, 180(12), 2452–2471. [doi:10.1016/j.cpc.2009.07.019]

PMI. (2013). *A Guide to the Project Management Body of Knowledge*, 5th Ed, PMI.

PRLIC, A., & Procter, J. B. (2012). Ten simple rules for the open development of scientific software. *PLoS computational biology*, 8(12), e1002802. [doi:10.1371/journal.pcbi.1002802]

PURSHOUSE, R. C., Ally, A. K., Brennan, A., Moyo, D., Norman, P. (2014). Evolutionary Parameter Estimation for a Theory of Planned Behaviour Microsimulation of Alcohol Consumption Dynamics in an English Birth Cohort 2003 to 2010. *GECCO '14: Proceedings of the 2014 Genetic and Evolutionary Computation Conference (in press)*. ACM.

READ, M., Timmis, J., Andrews, P. S., & Kumar, V. (2009). Using UML to model EAE and its regulatory network. In*Artificial Immune Systems* (pp. 4-6). Springer Berlin Heidelberg. [doi:10.1007/978-3-642-03246-2_2]

RIDOLFO, B. & Stevenson, C. (2001). The quantification of drug-caused mortality and morbidity in Australia 1998. Drugs Statistics Series. Australian Institute of Health and Welfare Canberra.

ROCHE, A. M., Pidd, K., Berry, J. G., & Harrison, J. E. (2008). Workers' drinking patterns: the impact on absenteeism in the Australian work-place. *Addiction*, 103(5), 738–748. [doi:10.1111/j.1360-0443.2008.02154.x]

ROOM, R., Osterberg, E., Ramstedt, M., & Rehm, J. (2009). Explaining change and stasis in alcohol consumption. *Addiction Research & Theory*, 17(6), 562–576. [doi:10.3109/16066350802626966]

ROOM, G. (2011). *Complexity, Institutions and Public Policy – Agile Decision-making in a Turbulent World*. Edward Elgar, Cheltenham.

SANDVE, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). Ten simple rules for reproducible computational research. *PLoS computational biology*, 9(10), e1003285. [doi:10.1371/journal.pcbi.1003285]

SALO, O., & Abrahamsson, P. (2004). Empirical evaluation of agile software development: The controlled case study approach. In *Product Focused Software Process Improvement* (pp. 408-423). Springer Berlin Heidelberg. [doi:10.1007/978-3-540-24659-6_29]

SCHILD, J., Walter, R., & Masuch, M. (2010, June). ABC-Sprints: adapting Scrum to academic game development courses. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games* (pp. 187–194). ACM.

SCHWABER, K. (1997). Scrum development process. In *Business Object Design and Implementation* (pp. 117–134). Springer London. [doi:10.1007/978-1-4471-0947-1_11]

SCHWABER, K. & Sutherland, J. (2013). The Scrum Guide™. *Scrum*. Scrum.org, July.

SINGLE, E., Robson, L., Xie, X., & Rehm, J. (1996). *The cost of substance abuse in Canada. Canadian Centre on Substance Abus*e, Ottawa.

SOMMERVILLE, I. (2009). *Software Engineering* (9th Edition). Addison Wesley.

University of Essex. Institute for Social and Economic Research (ISER). British Household Panel Survey: Waves 1-18, 1991–2009 [computer file]. 7th Edition. Colchester, Essex: UK Data Archive [distributor], July 2010. SN: 5151.

WILSON, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., ... & Wilson, P. (2014). Best practices for scientific computing. *PLoS biology*, 12(1), e1001745. [doi:10.1371/journal.pbio.1001745]