Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

SOFTWARE SKILLS REQUIRED BY M&S GRADUATES FOR DES DEVELOPMENT

James F. Leathrum, Jr.
John A. Sokolowski
Yuzhong Shen
Michel Audette

Department of Modeling, Simulation and Visualization Engineering
Old Dominion University
Norfolk, VA 23529, USA

ABSTRACT

The creation of a new modeling and simulation engineering program presented the opportunity to evaluate the core skills desirable in a well-rounded graduate. Software was identified as an often neglected aspect of modeling and simulation programs and an attempt was made to remedy this. This paper discusses the software skills identified as necessary/desirable in a graduate. The focus is on discrete event simulation (DES), though the skills are transferable to other paradigms. The discussion is partitioned into discussing skills appropriate for simulation application development and simulation tool development. The discussion is further partitioned to discuss core computer science skills (object-oriented programming and data structures), software architecture, graphics development, implementing DES worldviews, and an ability to work with open source software. The result is a graduate that is desirable to industry and graduate research.

1 INTRODUCTION

The creation of the first ABET accredited undergraduate modeling and simulation (M&S) engineering program at Old Dominion University (Mielke et al. 2011; Leathrum and Mielke 2012) presented the opportunity to evaluate required core competency in a graduate. An undergraduate program has the luxury of developing the necessary background skills to produce a well-rounded M&S professional. The result is a curriculum balanced between core M&S concepts, mathematics with an emphasis on analysis, and software development. The curriculum culminates in a year-long capstone experience where students develop a project from proposal to prototype for an industrial partner. The design of the curriculum came from a careful consideration of desired capabilities, drawing from the M&S Body of Knowledge (Tolk 2010), industrial partnerships, and greater than 10 years of experience from a graduate program in modeling and simulation engineering.

It was clear that teaching M&S purely from a tool-based perspective would be a disservice to the students. In order to address the needs of industry and research, a professional needs a strong software background. This allows the professional to develop simulation software beyond the scope and capabilities of existing simulation tools, develop the next generation of simulation tools, to interface existing simulations, and to interface with hardware (simulators, haptic devices, etc.).

Software development coursework begins with basic computer science courses in programming, object-oriented design, and discrete math. It then continues with software-intensive M&S courses in simulation software design and computer graphics. The development of these courses resulted in a modification to the existing graduate program to provide a single course to cover all of the software concepts covered in the undergraduate program, in a compressed format. This experience resulted in careful consideration of what content was most important to cover.

This paper captures the conclusions drawn from our experience in creating the program. The paper covers skills necessary for discrete event simulation, but most of the skills presented are transferable to other simulation paradigms. The paper focuses on two primary aspects of M&S software development: development of simulation applications in software and development of simulation tools. While there is significant overlap between the two, there are some skills unique to each making it appropriate to address them individually. The discussion highlights the software skills that we believe are important for a graduate. While the skills highlighted are considered of high importance, it should not be considered to downplay the importance of general software development skills to include software development and testing skills. It is assumed that these skills are obtained in general computer science coursework, and can then be assessed within the curriculum in courses such as a senior capstone course. The skills presented are covered within our curriculum to varying levels of depth. The skills then allow students to better succeed in following elective courses or graduate courses such as distributed simulation and agent-based simulation where they can now develop the simulation software instead of solely relying on tools and getting a precursory exposure to the underlying simulation issues.

2 APPLICATION DEVELOPMENT

2.1 Object-Oriented Programming

One of the main software development concepts practiced today is object-oriented programming (OOP). This concept grew out of a simulation language development effort in the 1960s (Dahl 2002). OOP has the software developer think about what he or she plans to develop in terms of objects that represent concepts or physical items, specifically how those objects behave and interact. This methodology mirrors how a modeling and simulation (M&S) engineer would approach developing a computer simulation. Modeling involves identifying items (objects) to be modeled and then making explicit the behaviors and relationships that are unique to each item. Consequently, OOP is a natural way for an M&S engineer to capture these concepts. Therefore, he or she should be schooled in OOP principles and an associated programming language that allows for developing M&S applications from scratch. This software development skill is a key element of Old Dominion University's approach to its M&S curriculum at both the undergraduate and graduate levels.

There are three distinguishing elements of OOP that set it apart from other software development methodologies. They are inheritance, polymorphism, and encapsulation (Chaudhari 2008). Inheritance allows an object to acquire properties of another object. This acquisition is done in a hierarchical manner. The M&S engineer saves time by reusing what they already developed and concentrating on what they need to add that makes a derived object unique from its parent. For example, we can define a vehicle object with certain universal characteristics found in most vehicles. Then we can define a car object that inherits the common vehicle properties as part of the car design.

Encapsulation binds together code and data related to a specific object and keeps both from misuse or unintentional modification. The M&S engineer, as the object designer, then specifies interfaces that allow access to the object and its functionality, while making sure that its structure is not unnecessarily modified.

Polymorphism, literally translated, means the ability to take more than one form. In terms of software design, it enables objects to have multiple properties, depending on how the object is used or instantiated. The M&S engineer specifies a general structure for an object and then can tailor it to specific purposes depending on the needs of the simulation. For instance, software can be developed to implement a graph topology to connect nodes using edges. The nodes can then be implemented as different process tasks and then interconnected to create a process, while the graph can simply view the individual process tasks as graph nodes.

OOP facilitates breaking down complex modeling and simulation problems into manageable parts, in the form of objects, to act as building blocks for multi-part systems that would be difficult to represent without this type of approach (Mota 2015). As models become more complex, this skill is necessary to master to be an effective M&S engineer. Researchers have developed libraries of software models to facilitate this process (Ward 1992).

2.2 Data Structures

Data structures play an important role in computer simulation. Organized data are often required as input to the simulation to help define the system being modeled (Bengtsson 2009; Allen 2016). Computer simulations also rely on internal data structures to capture system parameters and states as the simulation executes. One example is event organization in discrete event simulation (DES) (Schriber 1997). DES utilizes an event list to keep track of the proper order of events that must be executed by the simulation. Simulation engineers use one of several data structures to contain this list. Those structures include linked lists, various tree structures, and hash tables to mention a few. Implementation of these structures requires knowledge of how to efficiently code them in an OOP environment since their structure can account for significant computational load associated with DES. Simulation engineers must also understand the benefits and limitations of various data structures so that a proper method can be chosen for a particular application. Knowledge of lists, graphs, trees, and hashing algorithms is required to make the appropriate choice.

2.3 Programming DES Worldviews

Rashidi (2016) provides an assessment of current simulation software packages, categorizing them based on six taxonomies. He bases the majority of his taxonomies on the simulation development environment or higher level modeling perspectives. The focus of this discussion falls under Rashidi's first taxonomy: worldviews. There is plenty of literature covering worldviews such as (Overstreet and Nance 2004; Pegden 2010; Pidd 2004; Sargent 2004), but little exists in educational textbooks. Worldviews are defined as some subset of four approaches: event scheduling, activity scanning, three-phase, and process interaction (Balci 1988). The approaches differ in the representation of events and the underlying management of events. The two most prevalent approaches used in commercial software are event scheduling and process interaction (Rashidi 2016). In event scheduling, the system is viewed from the perspective of system state. Time-stamped events are ordered by time of occurrence. As the events occur over time, each event may change the system state and may schedule new future events. This worldview requires the modeler to express the system model in terms of events and states, which for many systems is known to be difficult (Pegden 2010). In process interaction, the system is viewed from the perspective of entities moving through one or more system processes. Since the late 1980's, process interaction has replaced event scheduling (Pegden 2010) as the most frequently used approach. Rashidi's survey of 62 simulation packages supports this claim, finding 32 software packages using process interaction, 26 using event scheduling, and a total of 20 using forms of the other two worldviews.

However, it is interesting to note that the overwhelming majority of available general simulation educational materials still focus on the event scheduling approach. This educational focus may be due to historical reasons (event scheduling was developed earlier), a consequence of the simulation name (it is Discrete Event Simulation, not Discrete Process Simulation), or the fact that process interaction representations can be converted to an event scheduling representation (Overstreet and Nance 2004). Two of the authors made the case previously (Leathrum et al. 2017) that both event scheduling and process interaction should be included in an M&S curriculum and proposed a unified view, finding the commonality between the two views, to avoid teaching them as distinctly different concepts when introducing software implementations.

2.3.1 Event Scheduling Worldview

The primary concept when developing software for the event scheduling worldview that is foreign to students with a classic computer science education is the need to schedule a call to an event subroutine to occur at a later execution time based on the scheduled simulation time. Texts such as (Nutaro 2011) take the approach of creating a subroutine which when provided an event id calls the appropriate event subroutine. However, this approach does not support reusable code and thus development of libraries to support the event scheduling worldview.

Instead, we teach students how to encapsulate an object method call for future execution. They are taught the concept of command patterns (Gamma et al. 1994) to encapsulate a specific event subroutine call to include the object, the method to be called, and a parameter list. This is consistent with the approach taken in libraries such as SIM (Boiler and Eliëns 1995) and SystemC (Open SystemC Initiative 2003). By studying how a simulation executive executes events defined as command patterns, students gain a greater appreciation of polymorphism and encapsulation, and should be better prepared to utilize a simulation executive library due to a better appreciation of its functionality and interaction with their application.

Experience has also shown that it is easier to teach DES software development if starting from a common modeling paradigm. (Leathrum et al. 2017) proposed the event graph (Schruben 1983) as the starting point as it clearly defines all event scheduling activities as well as the state changes. There is also literature showing how to transform other modeling paradigms to the event graph paradigm (Schruben and Yucesan 1994) and even other world views to event scheduling (Overstreet and Nance 2004).

2.3.2 Process Interaction Worldview

The process interaction worldview presents both advantages and disadvantages when introducing it to students. On the one hand, programming a single process is just an exercise in structured programming. However, students must have an appreciation for the concurrency of the interacting processes. While they may grasp this at the modeling level and when utilizing simulation tools, when stepping to software design this seems a more difficult concept. Students more readily map the sequential nature of event scheduling repeatedly selecting events for execution than transferring control back and forth between processes.

As in event scheduling, it is advantageous to start the software development process from a common modeling point. UML activity diagrams and/or statecharts are selected as an appropriate representation of the inherent concurrency (Crichton et al. 2001) that readily maps to software. Individual processes are defined in individual swimlanes. Each swimlane can then be developed as a routine implementing that process in software. Providing students with library support for process interaction and the passage of simulation time then allows them to focus on the development of the processes without worrying about the management of context switching between processes. Again, this approach readily sets up the students for using available process interaction capabilities in software libraries, in particular, SIM (Boiler and Eliëns 1995) and SystemC (Open SystemC Initiative 2003) both support both world views.

2.4 Open Source Software

The Internet has been the foundation of a revolution in how research and development are carried out, in making widely available subject matter expertise on a variety of spheres of activity, including YouTube-based lessons and broad dissemination through open publications such as FrontiersIn.org and arxiv.org (Frontiers 2019) (arXiv.org 2019). In parallel, a large number of software engineers have not only made available their expertise, but also disseminated their software implementation and data repositories, in areas ranging from computer visualization (VTK 2019) to physiological models (CellML 2019) (BioGears 2017). On occasion, academic publications also encourage contributors to make either their software code or a visualizable model available as an adjunct to their paper.

Equally important, these tools are also finding support in industry, where employers who view them as a means of quickly ramping up software projects, while exploiting complementary software tools that support software projects, such as GIT version control (Git 2019), Mantis bug tracker software (Mantis 2019), Doxygen documentation generation (Doxygen 2018) or CMake cross-platform compilation support (CMake 2019). It is not uncommon for job ads, such as those on Indeed.com (Indeed 2019), to specify as a hiring criterion the required expertise in many of the aforementioned open-source software implementations or software project support tools.

As educators, it has been apparent that we would be delinquent in preparing engineering students for the workforce if we neglected to account for the importance of the emergence of these open-source tools, in potentiating their toolbox as future M&S engineers. Moreover, learning to write software in an open source-aware manner is often significantly different than learning to write software in the absence of these tools. First of all, students have to become experts in reading code written by other people and discerning whether a specific implementation is appropriate for their needs. Moreover, these tools may require an important effort in building the publicly available software from source code, in that they are often reliant on other software libraries. If the build process of these open-source tools fails to conclude successfully, the software engineer must become an expert in these second- or third-party libraries to produce linkable libraries that can be integrated into his/her own code.

As a result, our department has encouraged students to consider open-source building blocks in their implementation, both for discrete event simulation classes as well as Capstone classes. In discrete event simulation, part of the emphasis is certainly on sophisticated Commercial-Off-The-Shelf (COTS) DES software, such as ARENA (Arena 2019) and SIMIO (Simio 2019), as it should be, especially early on in the program. As described in the survey by Dagkakis (Dagkakis 2016), the strength of such packages lies in the graphical tools for modeling, debugging and experimentation, provided to the user, combined with advanced visual facilities that enhance model development. However, the program is dedicated to producing M&S engineers, not merely users, so the junior and senior years encourage the use of high-level tools available in open source, in addition to software enabling tools such as GIT. In addition, commercial software tools are limited by cost, flexibility and reusability, whereby open-source DES is a compelling alternative (Dagkakis 2016).

There is in fact a plethora of open-source DES tools, and no single implementation seems to dominate currently. Dagkakis (2016) cites OMNeT++ (OMNeT++ 2019) , NS-3 (ns-3 2019), SimPy (SimPy 2018) and Jaam SIM (Jaam Sim 2019) in particular, with the latter deemed exceptionally promising in terms of offering functionality that competes with COTS software packages. Important considerations in the choice of open-source DES software include the specifics of the license and the language in which the package is implemented. In the former case, we strive to educate students on the broad license categories, and in particular make them aware of the subset of licenses with copy-left requirements that preclude the sale of commercial products built on them. As for the implementation language, many packages are developed in C++ or Python, which typically entails a trade-off of computational performance versus ease of implementation, which factors into their design choice.

Over the years, many Capstone projects have relied on Qt (Qt 2019) for their graphical user interface component, and an open-source counterpart under the hood as appropriate to the nature of the project. The customer varies every year, and the interests of this customer dictate the nature of the Capstone project and the choice of open-source components. Newport News Shipbuilding is particularly dedicated to the application of M&S in its fabrication and logistics processes; not surprisingly, this company has its own proprietary M&S software, which one senior group has leveraged as part of their Capstone project (Allen et al. 2014). More recently, a third group exploited Open MPI (Open MPI 2019) for a high-performance computing application that emphasized its Message Passing Interface. A fourth group exploited the Open Dynamics Engine (Smith 2019) in conjunction with a Navy project on naval simulation (Branch et al. 2017).

The current senior group (class of 2019) is exploiting SUMO traffic simulation (DLR - Institute of Transportation Systems 2019) as well as the TRACI TCP-based client-server interface to SUMO (TraCI 2019), to develop a proof-of-concept for their autonomous vehicle (AV) testbed based on simulated automobile navigation. The AV will be simulated by synthetic vehicle using AirSim (Microsoft 2019). The graphical user interface for this project is once more Qt-based.

2.5 Graphics

Visualization is an integral component of modeling and simulation and it is used in various stages of discrete event simulation (DES), such as input analysis, model development, results display, verification and validation, and analysis. Three levels of graphics software development capabilities are expected for

M&S graduates: 1) code development from scratch, 2) understanding and reuse of source code of existing projects or co-workers, and 3) efficient utilization of third-party software libraries when needed.

Most software applications involve different types of files and DES is no exception. File input and output are perhaps the first tasks to be addressed when developing DES applications for the real world. DES involves many types of files such as input files, output files, image files and even 3D files. M&S graduates are expected to understand relatively simple file structures and develop corresponding file readers and writers. For example, images are used in DES to represent different entities and statuses (e.g., busy, idle). The portable pixmap format (PPM) is a relatively simple image format and our students are required to write code from scratch to read and manipulate existing PPM files and generate new PPM files (screen captures). M&S graduates should understand file structures of moderate complexity. For instance, OBJ is a file format for representing polygonal meshes, including vertex positions, normals, colors, and texture coordinates. Our students are required to read, understand, and reuse the OBJ file parser developed by the instructor. For extremely complex or proprietary file formats, such as JPEG and FBX, students are expected to make full use of third-party software libraries to access these files in their code.

Although not necessarily mandatory for customized DES applications, graphical user interfaces (GUI) are a standard component of general DES tools such as Arena. Various building blocks allow users to create DES applications visually in a drag-and-drop manner and visualization of input, output, and other components enable users to understand and manipulate simulations effectively and efficiently. M&S graduates are required to generate primitive geometries procedurally (i.e., directly in the code, not pre-generated by an artist) and apply affine transformations (translation, rotation, and scaling) to produce multiple instances of the same base geometry. Also required are software implementations of complex mathematical functions, conversion between one-dimensional arrays to multi-dimensional arrays (e.g., 3-dimensional or 4-dimensional arrays), basic matrix operations (e.g., transpose and multiplication), and memory layout for variables and objects. Students should understand different visual representations of the same object and select the most effective one. For instance, while the length of a queue for a call center can be represented by a number inside a box, the queue for a restaurant or a bank can be visualized by icons or 3D models representing the customers, offering the user direct appreciation of the physical restraints (space) on the maximum queue length. Understanding of different spaces, such as object space, world space, camera space, and screen space, and conversions among different spaces is critical for creating correct visualizations. Utilization of camera projections (orthographic and perspective), viewport, lighting, shading, and texture mapping is required to produce an informative visualization that allows the application user to understand the simulation from multiple perspectives.

M&S students are required to develop various data structures (structures, enumerations, and classes) to store graphical entities such as geometry (vertex positions), topology (triangle, polygon), and visual attributes (colors and texture coordinates). Utilization of the classes such as *vector* and *map* provided by the C++ Standard Template Library (STL) allow for rapid application development. Students should be able to develop reusable code via inheritance and containment/delegation. Students are expected to design classes to properly model real-world objects with the understanding the same object may need multiple representations, such as behavior model, graphical model, or even physics model for a realistic 3D rendering of a DES application. Students need to create class hierarchies by applying the key principles of OOP programming (inheritance, encapsulation, and polymorphism) and advanced classes that contain other more basic classes using the containment/delegation model. These software design and development principles and techniques are applied to the entire application development to produce an overall software architecture that is clean and adaptive to future revision and upgrades. This perspective clearly supports the previous discussion on the importance of object-oriented programming.

Another important aspect is the utilization of industry standard software library tools. OpenGL is a cross-platform application programming language (API) for 3D computer graphics and it is the de facto industry standard for professional graphics, such as architecture design, scientific visualization, and information visualization (OpenGL 2019). It is available on Microsoft Windows, Linux, MacOS, Android, iOS, and game consoles. Utilizing a low-level library like OpenGL in instruction ensures students have deep understandings of the theories of computer graphics and visualization, providing students a solid foundation to build their visualization capabilities on top of OpenGL or make use of other high-level libraries, such as OpenSceneGraph.

3 SIMULATION TOOL DEVELOPMENT

3.1 Software Architecture

The nature of simulation software development requires solid architecture design to enable both reusability and extensibility. Students should be exposed to the main concepts, and be comfortable with at least two basic architecture models. The first is directly relevant to development of simulation libraries and important in tool development, the concept of separating the application from the simulation executive. (Pidd 2004) presented this simple decomposition where the application schedules events with the simulation executive and the simulation executive executes events on the application.

In addition, it is important for students to understand the interrelationship between components in the system, especially between the core simulation and the user interface. They should be exposed to reusable *design patterns* such as Model-View-Controller (MVC) and Model-View-Presenter (MVP) (Qureshi and Sabir 2013). This instruction on design patterns assists them in understanding the decomposition at various levels of the software design.

3.2 Simulation Executive Development

A relatively unique experience of the students in ODU's program is their exposure to the inner workings of the simulation executive. They not only understand its inner workings and why it matters (Schriber 2014), but also the implications of software design decisions. These foundations are highlighted in three areas: data structure design of the event list, the basic control structure of the executive, and a basic understanding of the mechanisms to execute an event in both the event scheduling and process interaction worldviews.

When discussing the event list, it is a perfect opportunity to demonstrate to students the impact of selecting different data structures for implementation. They can start with a simple linked list solution, but then the class quickly introduces alternative solutions using hash tables and binary search trees to enable students to better understand their potential benefits and pitfalls. This material leads into further advanced solutions such as calendar queues (Brown 1988), lazy queues (Rönngren 1991) and ladder queues (Tang et al. 2005), in a never ending search for an O(1) solution. This class is not intended to promote one solution over another, but rather to expose the student to options and design considerations.

Finally, the student is exposed to the different ways to execute an event based on the worldview. They are fully exposed to the event scheduling approach, and are capable to execute events in the simulation executive with no knowledge of the application's event structure thanks to the use of *command patterns*. Students become so comfortable with this approach that it becomes their fallback technique when posed with an industry driven problem in their Capstone course, opting to build their own (or rather use the one built in a prior simulation software design course) over utilizing a simulation software library. Process interaction poses a more difficult problem when presenting approaches to implementation. Students are not exposed to multithreading in the current computer science curriculum, let alone the preferred co-routine approach (Weatherly and Page 2004; Xu and Li 2012). So they have to settle with a blackboard discussion of the approach. They are provided a homegrown library implemented using threads, threads chosen due to their inclusion in the C++11 standard alleviating the need to use further libraries despite the loss in performance.

4 CONCLUSIONS

This paper presents the core software skills that we believe are necessary for a well-rounded simulationist. While not all practitioners need a thorough background in these skills, we argue that students should minimally be aware of them. As such, our undergraduate program provides a fairly thorough coverage with some exceptions such as the implementation of the process interaction worldview. Graduate students come from varied backgrounds, and do not have the time in the program to develop these skills. However, we have created a graduate-level course that covers all of these topics in a single semester, giving students exposure, albeit little time to practice the skills necessary to become competent. Our graduates are valued for their breadth in the field, adept at core M&S, analysis, and software development: an ability that more than one employer has commented as being an attractive quality in a candidate. Students have also been more productive in graduate research as many projects require software development. Therefore, we believe our efforts in including core software skills in a M&S curriculum have been well founded.

The original curriculum design involved identifying core skills required by an M&S professional based on various sources such as the M&S Body of Knowledge (Mielke et al. 2011; Leathrum and Mielke 2012). By delving deeper into the high-level concepts down to the level described in this paper, available university courses, in particular from the Computer Science department, can be organized into a curriculum. But it also highlights where the program's needs go beyond concepts available in general software courses.

REFERENCES

Allen, A., J. Caldwell, C. Heard, I. Sakiotis, and D. Tillinghast. 2014. "Discrete Event Simulation for Supporting Production Planning and Scheduling Decision in Job Shop Facilities". In *MODSIM World 2014*, April 15th-17th, Virginia Beach, VA, 444-448.

Allen, G. 2016. "Modeling and Simulation Data Integration - Inviting Complexity," *Journal of Cyber Security and Information Systems* 4(2):2-6.

Arena. 2019. Arena Simulation Software. https://www.arenasimulation.com/, accessed 3rd April 2019.

arXiv.org. 2019. Cornell University. arxiv.org, accessed 3rd April 2019.

Balci, O. 1988. "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages". In *1988 Winter Simulation Conference Proceedings*, edited by M. Abrams, P. Haigh, and J. Comfort, 287-295. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc..

Bengtsson, N., G. Shao, and B. Johansson. 2009. "Input Data Management Methodology for Discrete Event Simulation," in *Proceedings of the 2009 Winter Simulation Conference*, edited by M. Rossetti, R. Hill, B. Hohansson, A. Dunkin, and R. Ingalls, 1335-1344. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc..

BioGears. 2017. What Can BioGears Do? biogearsengine.com, accessed 3rd April 2019.

Bolier, D. and A. Eliëns. 1995. Sim: A C++ Library for Discrete Event Simulation. http://www.cs.vu.nl/~eliens/sim/. accessed 31st March 2019.

Branch, B., S. Collins, L. Dumaliang, N. Gonda, T. Lane, K. Miles, M. Periman, and D. Scerbo. 2017. "Rapid USV Prototyping System (RUPS)". In *MODSIM World 2017*, April 27th-29th, Virginia Beach, VA.

Brown, R. 1988. "Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem." *Communications of the ACM*. 31(10):1220-1227.

CellML. 2019. The CellML Project. cellml.org, accessed 3rd April 2019.

Chaudhari, S. 2008. Object Oriented Programming Concepts. https://www.codeproject.com/Articles/27775/Object-Oriented-Programming-Concepts, accessed 28th March 2019.

Crichton, C., J. Davies and A. Cavarra. 2001. *A Pattern for Concurrency in UML*. Technical Report PRG-RR-01-22. Oxford University Computing Laboratory, Oxford University, Oxford, UK. https://www.cs.ox.ac.uk/techreports/oucl/RR-01-22.pdf, accessed 31st March 2019.

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

Dahl, O. J. 2002. "The Roots of object Orientation: the Simula Language". in *Software Pioneers: Contributions to Software Engineering, Programming, Software Engineering and Operating Systems Series*, edited by M. Broy and E. Denert, 78-90. Berlin, Germany: Springer-Verlag.

DLR - Institute of Transportation Systems. (2019). SUMO - Simulation of Urban MObility. https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/, accessed 3rd April 2019.

Doxygen. 2018. *Generate documentation from source code*. http://www.doxygen.nl/, accessed 3rd April 2019.

Frontiers. 2019. Frontiers - 10 Years. https://www.frontiersin.org/, accessed 3rd April 2019.

Gamma, E, R. Helm, R. Johnson, and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley Professional.

Git. 2019. Local Branching on-the-cheap. https://git-scm.com/, accessed 3rd April 2019.

Indeed. 2019. Indeed. https://www.indeed.com/, accessed 3rd April 2019.

Jaam Sim. 2019. Leading Edge Simulation. https://jaamsim.com/, accessed 3rd April 2019.

Leathrum, J. and R. Mielke. 2012. “Outcome-Based Curriculum Development for an Undergraduate M&S Program”. In *AutumnSim 2012, Conference on Education and Training Modeling and Simulation (ETMS’12)*, edited by A. Abhari, 48-53. San Diego: Society for Modeling & Simulation International.

Leathrum, J., R. Mielke, A. Collins, and M. Audette. 2017. “Proposed Unified Discrete Event Simulation Content Roadmap for M&S Curricula”. In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. Chan, A. D’Amborgio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 4300-4311. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mantis. 2019. Mantis Bug Tracker. https://www.mantisbt.org/, accessed 3rd April 2019.

Microsoft. 2019. Aerial Informatics and Robotics Platform. https://www.microsoft.com/en-us/research/project/aerial-informatics-robotics-platform/, accessed 3rd April 2019.

Mielke, R., J. Leathrum, and F. McKenzie. 2011. “A Model for University-Level Education Modeling and Simulation”. *M&S Journal*, 6(3):14-23.

Mota, F. A., J. N. Hinckel, E. M. Rocco, E. M. and H. Schlingloff. 2015. "Modeling and Simulation of Launch Vehicles Using Object-Oriented Programming: Impact of the Engine Paramters on the Launcher Performance". In *Proceedings of the 6th European Conference for Aeronautics and Space Sciences (EUCASS 2015)*, June 29th-July 3rd, Krakow, Poland.

ns-3. 2019. Network Simulator. https://www.nsnam.org/, accessed 3rd April 2019.

Nutaro, J. 2011. *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Hoboken, New Jersey: John Wiley & Sons, Inc.

OMNeT++. 2019. Discrete Event Simulator. https://omnetpp.org/, accessed 3rd April 2019.

Open MPI. 2019. Open Source High Performance Computing. https://www.open-mpi.org/, accessed 3rd April 2019.

Open SystemC Initiative. 2003. SystemC 2.0.1 Language Reference Manual. http://homes.di.unimi.it/~pedersini/AD/SystemC_v201_LRM.pdf. Accessed 31st March 2019.

OpenGL. 2019. OpenGL: The Industry’s Foundation for High Performance Graphics. https://www.opengl.org/. Accessed 10th April 2019.

OpenSceneGraph. 2019. The OpenSceneGraph Project Website. http://www.openscenegraph.org/. Accessed 10th April 2019.

Overstreet, C. and R. Nance. 2004. “Characterizations and Relationships of World Views.” In *Proceedings of the 2004 Winter Simulation Conference*, edited by R. Ingalls, M. Rossetti, J. Smith, and B. Peters, 279-287. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc..

Pegden, C. D. 2010. “Advanced Tutorial: Overview of Simulation World Views”. In *Proceedings of the 2010 Winter Simulation Conference,* edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 210-215. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc..

Pidd, M. 2004. “Simulation Worldviews – So What?”. In *Proceedings of the 2004 Winter Simulation Conference*, edited by by R. Ingalls, M. Rossetti, J. Smith, and B. Peters, 288-292. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Qt. 2019. Software development made smarter. https://www.qt.io/, accessed 3rd April 2019.

Qureshi, M. and M. Sabir. 2013. “A Comparison of Model View Controller and Model View Presenter.” *Science International-Lahore*, 25(1):7-9.

Rashidi, H. 2016. "Discrete Simulation Software: a Survey on Taxonomies". *Journal of Simulation*, 11(2):174-184.

Ronngren, R., J. Riboe, and R. Ayani. 1991. “Lazy Queue: An Efficient Implementation of the Pending-Event Set,” *ACM SIGSIM Simulation Digest*. 23(3):194-204.

Sargent, R. 1988. “Event Graph Modelling for Simulation with an Application to Flexible Manufacturing Systems”. *Management Science*. 34(10):1231-1251.

Sargent, R. 2004. “Some Recent Advances in the Process World View”. In *Proceedings of the 2004 Winter Simulation Conference*, edited by by R. Ingalls, M. Rossetti, J. Smith, and B. Peters, 293-299. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schriber, T. J. and D. T. Brunner. 1997. “Inside Discrete-Event Simulation Software: How It Works and Why it Matters”. In *Proceedings of the 1997 Winter Simulation Conference*, edited by S. Andradóttir, K. Healy, D. Withers, and B. Nelson, 14-22. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schriber, T., D. Brunner, and J. Smith. 2014. “Inside Discrete-Event Simulation Software: How It Works and Why It Matters”. In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Diallo, I. Ryzhov, L Yilmaz, S. Buckley, and J. Miller, 132-146. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schruben, L. 1983. “Simulation Modeling with Event Graphs”. *Communications of the ACM*, 26(11):957-963.

Schruben, L. and E. Yucesan. 1994. “Transforming Petri Nets into Event Graph Models”. In *Proceedings of the 1994 Winter Simulation Conference*, edited by J. Tew, S. Manivannan, D. Sadowski, and A. Seila, 560-565. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schruben, L.. 1995. “Building Reusable Simulators using Hierarchical Event Graphs”. In *Proceedings of the 1995 Winter Simulation Conference*, edited by C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, 472-475. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Simio. 2019. Forward Thinking. https://www.simio.com/index.php, accessed 3rd April 2019.

SimPy. 2018. Discrete event simulation for Python. https://simpy.readthedocs.io/en/latest/, accessed 3rd April 2019.

Smith R. 2019. Open Dynamics Engine. http://ode.org/, accessed 3rd April 2019.

Tang, W., R. Goh, and I. Thing. 2005. "Ladder Queue: An O(1) Priority Queue Structure for Large-Scale Discrete Event Simulation." *ACM Transactions on Modeling and Computer Simulation*. 15(3):175-204.

Tolk, Andreas. 2010. "M&S Body of Knowledge: Progress Report and Look Ahead". *SCS M&S Magazine*. 4:1-5.

TraCI. 2019. Sumo Wiki. https://sumo.dlr.de/wiki/TraCI, accessed 3rd April 2019.

VTK. 2019. Visualization Toolkit. vtk.org, accessed 3rd April 2019.

Ward, R. L. and W. V. Huang. 1992. "Simulation with Object Oriented Programming" *Computers & Industrial Engineering,* 23(1):219-222.

Weatherly, R. and E. Page. 2004. "Efficient Process Interaction Simulation in Java: Implementing Co-Routines within a Single Java Thread". In *Proceedings of the 2004 Winter Simulation Conference*, edited by by R. Ingalls, M. Rossetti, J. Smith, and B. Peters,, 1437-1443. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Xu, X. and G. Li. 2012. "Research on Coroutine-Based Process Interaction Simulation Mechanism in C++". In *AsiaSim 2012 - Asia Simulation Conference 2012 Proceedings, Part III (Communication in Computer and Information Science 325)*, edited by T. Xiao, L. Zhang, and M. Fei, 178-187. New York City, New York: Springer Publishing Company.

AUTHOR BIOGRAPHIES

JAMES LEATHRUM is an Associate Professor in the Department of Modeling, Simulation and Visualization Engineering at Old Dominion University. He earned the Ph.D. in Electrical Engineering from Duke University. His research interests include simulation software design, distributed simulation, and simulation education. His e-mail address is jleathru@odu.edu.

JOHN SOKOLOWSKI is an Associate Professor in the Department of Modeling, Simulation and Visualization Engineering at Old Dominion University. He is Chairman of the Board of the Society for Modeling and Simulation International and serves on the board of the Winter Simulation Conference. He is the previous Executive Director of the Virginia Modeling, Analysis and Simulation Center at ODU. His research interests are in the computational representation of human behavior in simulation. His email address is jsokolow@odu.edu.

YUZHONG SHEN is a Professor in the Department of Modeling, Simulation and Visualization Engineering at Old Dominion University. He earned the Ph.D. in Electrical Engineering from the University of Delaware. His research interests include visualization and computer graphics, modeling and simulation, and signal and image processing. His e-mail address is yshen@odu.edu.

MICHEL AUDETTE is an Associate Professor of the Department of Modeling, Simulation and Visualization Engineering at Old Dominion University. He holds a Ph.D. in Biomedical Engineering from McGill University. His research interests include medical simulation, featuring physiological state-based virtual-reality simulation, surgery planning and medical image analysis. His e-mail address is maudette@odu.edu.