Journal of Information Technology Management

ISSN #1042-1319

A Publication of the Association of Management

A HEURISTIC METHOD FOR SCHEDULING REQUIREMENTS IMPLEMENTATION IN AGILE SOFTWARE DEVELOPMENT PROJECTS

MARK L. GILLENSON
UNIVERSITY OF MEMPHIS
mgillnsn@memphis.edu

MICHAEL J. RACER
UNIVERSITY OF MEMPHIS
mracer@memphis.edu

XIHUI ZHANG
UNIVERSITY OF NORTH ALABAMA
xzhang6@una.edu

RUBY E. BOOTH
UNIVERSITY OF MEMPHIS
rbooth@memphis.edu

JOHN P. DUGAN
COROUTINE, LLC
jdugan@coroutine.com

ABSTRACT

One of the challenges of successfully developing software using an agile software development methodology is the determination of which requirements or features to implement in which of the agile iterations. These decisions involve priorities, risk, development costs, and testing costs. Furthermore, initial decisions may have to be revised on the fly during development due to the possibility, embraced by the agile development philosophy, of changes to the set of requirements, and to the possibility of software failure during certain aspects of testing that require code fixes before further development can continue. With a sample project, this paper presents a heuristic method for scheduling requirements implementation in agile development iterations, taking into account all of the aforementioned considerations.

Keywords: heuristic method, scheduling requirements implementation, agile development, software development and testing

INTRODUCTION

Agile software development is the culmination of a variety of rapid application development techniques that arose in response to the well-known problems of traditional "waterfall" software development [11] [21]. These problems include overly long development timelines, a lack of even a partial result to evaluate until the entire project is completed, an intolerance for changing or additional requirements, and cumbersome testing that often does not commence until the project is completed [1] [4]. Furthermore, traditional software development typically separates the developers from the end-users through most of the development process, which runs counter to today's philosophy of closely integrating the "business people" with the information technology staff. As a result of all of these issues, software has often been late (if completed at all), has often not matched the stated requirements, and has contained an unacceptable number of defects [22] [32].

There are a variety of agile software development methodologies, some of the best known of which include scrum, extreme programming (XP), lean software development (LSD), agile unified process (AUP), feature-driven development (FDD), crystal methodologies, and dynamic systems development method (DSDM) [11]. These generally include several common "agile development" practices. There is a small, cross-functional development team that includes programmers, testers, other IT professionals as needed, and, very importantly, a user representative, who work together in "iterations" that typically last one to four weeks, depending on the particular methodology employed [9] [19]. Development begins with the "critical path" or central feature of the application and then adds other features as it progresses, "continuously integrating" additional features to progressively build the application [12]. In this way, from early in development, a semblance of the application can be evaluated and tested. Requirements can be modified along the way, new requirements can be added, and the user representative is always there to verify and answer questions about the requirements.

A challenge in agile development that has not been comprehensively addressed in the information technology literature is the scheduling of the implementation of application requirements or features across the series of iterations that comprise the agile development timeline. A carefully constructed agile implementation schedule should include elements of feature priority, risk involved of the feature not working properly, the cost of developing the feature, and, very importantly, the cost of testing, which is often not sufficiently taken into account [10]. Generally speaking, features of high priority (including the critical path feature) should be developed first, subject also to the degree of risk involved, to the development cost, and to the testing cost. Furthermore, the cost of testing is related to risk, as it stands to reason that a riskier element of software should be more thoroughly tested. There can also be mandatory orderings among some of the requirements due to required sequencing of feature development. Finally, any such scheduling scheme must be dynamic in nature as it must be able to continually adjust for software failures discovered by testing and by the addition of new features during development. We propose a new and comprehensive agile feature development scheduling procedure that takes all of these issues into account, and breaks new ground.

LITERATURE REVIEW

There are three distinct bodies of literature that relate to the problem of scheduling requirements and features for implementation and testing in agile development iterations: the body of literature that describes agile software development methodologies, the literature on software testing in general, and the discipline known as "scheduling" which in turn is considered to be a component of the field of operations management.

A considerable amount of agile development practice has already taken place and a considerable amount of literature has already been written about it, dating back over a twenty-year period [19] [21] [22] [27] [29]. All of these references describe the concept of the "iteration," which is central to agile software development and to the subject of this paper. A more recent survey of agile software development methodologies is found in [11]. A discussion of the importance of "daily testing" and its cost-reducing benefits, which is directly related to the subject of this paper, is found in [20].

The literature on software testing goes back at least as far as the classic *The art of software testing* [23]. Some of the software testing literature is purely technical in nature (e.g., [1] [7] [16]). Some of the literature is more managerial in nature (e.g., [6] [8] [15]). And some of the literature encompasses both technical and managerial aspects (e.g., [14] [24]). An interesting book that describes the software testing philosophy and methodologies of one company is authored by Whittaker et al. [32].

Further, the two fields of agile software development and software testing converged with the unique book *Agile testing: A practical guide for testers and agile teams* [9] and its sequel, *More agile testing: Learning journeys for the whole team* [12].

The classic, comprehensive reference on scheduling problems is [25], while more compact discussions on the topic are found in [26] and [28]. "An update" that provides a good overview of scheduling heuristics is [17]. Other references on scheduling will be cited in the sections below where they relate to our methodology.

A HEURISTIC METHOD

Much of the literature on scheduling, specifically "resource-constrained project scheduling" (e.g., [2] [30] [31]) considers specific problems for which specific solutions are developed (e.g., [5], which discusses scheduling tests in automotive development projects). We believe that our procedure for scheduling requirements and features among the iterations of an agile development effort is equally unique and has not been addressed in the literature to date. However, we have found elements of development methodologies in the scheduling literature that, while they are not based on the problem of software development, do relate to our problem in interesting ways. An exception, which does discuss scheduling and inspection in software development projects (but, not in agile software development projects, which limits its value to us) is [13]. We shall cite these papers where appropriate.

Consider that there is a set of requirements labeled *A-X*, each of which is to be implemented as a software module. In the agile development environment, consider that there are n iterations, each of which has a development capacity of *c* person days. That is, a certain amount of time has been allotted for the application development effort and therefore there are a limited number of available iterations, each with the same capacity. Each requirement is assigned a relative priority, *p*, on a scale of 1-5 where 1 is the highest priority and 5 is the lowest (see [5] for discussions of ordering tasks based on priorities). Each of the requirements (and its associated software module) is assigned a relative risk, *r*, on a scale of 1-5 where 1 is the lowest risk and 5 is the highest. There is also the possibility of mandatory orderings among the requirements where one requirement must be implemented before another, regardless of other considerations. This can happen, for example, when it is more convenient to test a particular software module after another has been completed.

Each requirement is assigned an estimated development cost in person days, *d*, which in the spirit of the agile development environment includes both coding and initial testing as the code is being developed. In this model, as is the case in many agile development environments, there is an additional level of testing with a testing cost, *t*. This additional level of testing could be comprised of further testing by the developers and testers on the agile team, could be a level of user acceptance testing, or could be a combination of the two. The additional testing cost, *t*, can be established on the same linear scale as the risk, *r*, or can be a non-linear factor of the risk, i.e., a riskier software module might require *much more* testing than a less risky module. Henceforth, when we refer to "testing," we are referring to this additional level of testing. Finally, the estimated total cost of developing and testing a software module, *d+t*, is *dt*.

Some modules will be developed and tested in the same iteration. Any necessary rework will be done in the next iteration. Some modules will be developed in iteration *x*, tested in iteration *x+1*, and have any necessary rework done in iteration *x+2*. Also, in the agile software development paradigm, new requirements may be introduced at any time during development.

The procedure begins with creating a requirements matrix by listing the requirements in order by their *A-X* labels. Each requirement is listed with its development cost, *d*, its priority, *p*, its risk, *r*, its (additional) testing cost, *t*, and its total cost, *dt*. Next, the rows of the matrix are reordered by highest to lowest priority, within equal priority by lowest to highest risk, and within equal priority and risk by lowest total cost. This strategy prioritizes the most important items that are most likely to survive the testing process and actually make it to production within the given timeframe.

For the first iteration and for every successive iteration, the remaining requirements in the requirements matrix and their associated software modules are assigned to the iteration in priority order, subject to the capacity of the iteration. If more than one of the highest priority requirements remain, the one that will fit within the remaining capacity of the iteration will be chosen. If the remaining capacity will accommodate the development but not the testing of the next requirement in line, the development will be done in this iteration and the testing will be held over to the next iteration. In this latter case, the testing of this software module will be first in line in assignment to the next iteration.

If a requirement is developed and successfully tested in an iteration, it is removed from the requirements matrix. If only the development was performed in a given iteration, it remains in the requirements matrix but with a development cost of zero and a recalculated total cost (which is equal to the testing cost, since that is the only cost remaining for that requirement).

If a requirement fails its test and thus requires more development time to fix whatever caused it to fail its test, it remains in the requirements matrix with a new estimated development cost and a new estimated testing

cost. Lambrechts et al. [18] consider changes to project scheduling due to failures in component testing (not related to software development). Ashtiani et al. [3] consider mid-project schedule alterations due to changes in resource needs.

If at any time a new requirement is added to the project, it is inserted into the requirements matrix in its proper priority, risk, and total cost order. The procedure iterates until either all of the requirements have been satisfied with successfully tested software modules, or the process has run out of iterations (and thus time) to work with.

A SAMPLE AGILE SOFTWARE DEVELOPMENT PROJECT

First, we present the content of the system and the basic definition of those units to be scheduled.

- 10 requirements *A-J*
- 10 software modules to be developed
- 5 iterations planned
- Capacity of each iteration: 16 person days (For the sake of clarity in the description, we will get as close as possible to the 16 person day limit in each iteration without resorting to fractional days or costs.)
- Each requirement has:
  - Development cost and testing cost in person days
  - Priority, 1-5, 1 is highest, 5 is lowest
  - Risk, 1-5, 1 is lowest, 5 is highest
  - Testing cost is directly related to risk
  - Possible mandatory ordering among requirements

We will consider the following properties to influence the decision-making process.

- Some modules will be developed and tested in the same iteration. Any necessary rework will be done in the next iteration.
- Some modules will be developed in iteration n, tested in iteration *n+1*, and have any necessary rework done in iteration *n+2*.
- New requirements may be introduced during development.

Table 1: Ten Initial Modules to Be Developed and Tested

Requirement	Development Cost	Priority	Risk	Testing Cost	Total Cost	Mandatory Ordering
A	3	1	4	4	7	A must be in Iteration 1
B	2	4	5	5	7	
C	1	3	2	2	3	
D	4	2	3	3	7	
E	4	4	5	5	9	E must be done before G
F	2	1	2	2	4	F must be done before G & H
G	2	5	1	1	3	
H	5	2	1	1	6	
I	4	3	3	3	7	
J	2	5	4	4	6	

Before starting the iterations, we need to sort the matrix the following way:

- Sort the rows of the matrix by highest priority, then lowest risk, and finally lowest total cost.
- This strategy prioritizes the most important items that are most likely to survive the testing process and actually make it to production within the given timeframe.
- However, the capacity remaining in an iteration may cause ordering changes.

Table 2: Ten Initial Modules (Sorted) to Be Developed and Tested

Requirement	Development Cost	Priority	Risk	Testing Cost	Total Cost	Mandatory Ordering
F	2	1	2	2	4	F must be done before G & H
A	3	1	4	4	7	A must be in Iteration 1
H	5	2	1	1	6	
D	4	2	3	3	7	
C	1	3	2	2	3	
I	4	3	3	3	7	
B	2	4	5	5	7	
E	4	4	5	5	9	E must be done before G
G	2	5	1	1	3	
J	2	5	4	4	6	

Iteration 1

- Iteration 1, begins by implementing the requirements with Priority 1, Requirements A and F.
- Both their development and testing will be performed in Iteration 1.
- Their combined total cost is 11, leaving another 5 person-days of capacity in Iteration 1.
- There are two requirements with Priority 2, Requirements D and H.
- D's total cost of 7 exceeds the remaining capacity in Iteration 1 and so does H's total cost of 6.
- So, a decision is made to perform only the development but not the testing of H (H is lower in risk than D) in Iteration 1 at a cost of 5 person-days.

Table 3: Tasks Performed in Iteration 1

Iteration 1
A 7
F 4
H 5

Development and testing conducted on the code for Requirements A and F during Iteration 1 and possibly right after Iteration 1 were successful.

- The requirements matrix going forward no longer includes Requirements A and F.
- Since the code for Requirement H was developed but not tested, it remains in the matrix but with a development cost of 0.

Table 4: Eight Modules to Be Developed and Tested before Iteration 2

Requirement	Development Cost	Priority	Risk	Testing Cost	Total Cost	Mandatory Ordering
H	0	2	1	1	1	
D	4	2	3	3	7	
C	1	3	2	2	3	
I	4	3	3	3	7	
B	2	4	5	5	7	
E	4	4	5	5	9	E must be done before G
G	2	5	1	1	3	
J	2	5	4	4	6	

Iteration 2

- Iteration 2 begins with the testing of Requirement H.
- Then it picks up the remaining Priority 2 requirement, Requirement D, for development and testing.
- The total capacity needed is 8 person-days, leaving 8 person-days of additional capacity.
- The Priority 3 requirements are C and I but only Requirement C, with a total cost of 3 for development and testing, will fit in the remaining capacity of Iteration 2 and so it is included. Note that Requirement C is lower in risk than Requirement I.
- Finally, only the development of Requirement I is included in Iteration 2 due to the limitation of the remaining capacity in the iteration.

Table 5: Tasks Performed in Iterations 1 and 2

Iteration 1	Iteration 2
A 7	H 1
F 4	D 7
H 5	C 3
	I 4

- After Iteration 2, testing indicated that the code for Requirement C needed 1 more person-day of development time to fix discovered defects and 1 more person-day of testing. Also, the testing of Requirement H indicated that it needed more work: 2 more person-days of development and 2 more days of testing.
- In addition, a new requirement, Requirement K, with Priority 1, was introduced into the project.
- Since Requirement I was developed but not tested in Iteration 2, it now appears in the matrix with development cost 0.

Table 6: Eight Modules to Be Developed and Tested before Iteration 3

Requirement	Development Cost	Priority	Risk	Testing Cost	Total Cost	Mandatory Ordering
K	3	1	2	2	5	
H	2	2	1	2	4	
C	1	3	2	1	2	
I	0	3	3	3	3	
B	2	4	5	5	7	
E	4	4	5	5	9	E must be done before G
G	2	5	1	1	3	
J	2	5	4	4	6	

Iteration 3

- Since the new Requirement K is Priority 1, its code is developed and tested in the next iteration, Iteration 3.
- The rework on Requirement H is performed and so is the rework on Requirement C.
- Finally, the testing of Requirement I, which was developed but not tested in the previous iteration is performed in Iteration 3.

Table 7: Tasks Performed in Iterations 1, 2, 3

Iteration 1	Iteration 2	Iteration 3
A 7	H 1	K 5
F 4	D 7	H 4
H 5	C 3	C 2
	I 4	I 3

- Iteration 3 successfully completed the development (or rework) and testing of the code for Requirements K, H, and C.
- The testing of the code for Requirement I indicated that it needed more work: 2 more person-days of development and 2 more person-days of testing.

Table 8: Five Modules to Be Developed and Tested before Iteration 4

Requirement	Development Cost	Priority	Risk	Testing Cost	Total Cost	Mandatory Ordering
I	2	3	3	2	4	
B	2	4	5	5	7	
E	4	4	5	5	9	E must be done before G
G	2	5	1	1	3	
J	2	5	4	4	6	

Iteration 4

- Iteration 4 will begin with the needed rework of Requirement I.
- It will also include the development and testing of Requirement B.
- Because of the stated mandatory ordering between Requirements E and G (plus the higher priority of Requirement E), and the limited capacity left in Iteration 4, the development of the code for Requirement E will be done in Iteration 4, but not its testing as Iteration 4 does not have enough remaining capacity for Requirement E's testing.

Table 9: Tasks Performed in Iterations 1, 2, 3, and 4

Iteration 1	Iteration 2	Iteration 3	Iteration 4
A 7	H 1	K 5	I 4
F 4	D 7	H 4	B 7
H 5	C 3	C 2	E 4
	I 4	I 3	

- The code for Requirement I was tested successfully and so it was completed.
- The code for Requirement E must still be tested, since it was only developed in the previous iteration.
- The testing of Requirement B indicated that it needed more work: 2 more person-days of development and 3 more person-days of testing.

Table 10: Four Modules to Be Developed and Tested before Iteration 5

Requirement	Development Cost	Priority	Risk	Testing Cost	Total Cost	Mandatory Ordering
B	2	4	5	3	5	
E	0	4	5	5	5	E must be done before G
G	2	5	1	1	3	
J	2	5	4	4	6	

Iteration 5

- Iteration 5 will include the rework of the code for Requirement B.
- The testing of Requirement E.
- The development and testing of Requirement G.

Table 11: Tasks Performed in Iterations 1, 2, 3, 4, and 5

Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
A 7	H 1	K 5	I 4	B 5
F 4	D 7	H 4	B 7	E 5
H 5	C 3	C 2	E 4	G 3
	I 4	I 3		

- Requirements B, E, and G, were successfully tested in Iteration 5 and so are completed.
- Since only 5 iterations were planned, it was decided that the development and testing of Requirement J will be held over for a later release.

CONCLUSION

When new information systems technologies are introduced, companies tend to "try them out" on small, low-risk projects, in effect as experiments. Those technologies that show promise are gradually rolled-out into mainstream projects. A good example of this was the introduction of relational database management in around 1980. Once experimental, it is now the technology of choice for virtually all new transactional system development and is also heavily used in decision support environments. In a similar fashion, the use of agile software development began cautiously but appears to be gaining ground rapidly, especially for smaller application development projects and, to some extent experimentally, for larger ones.

A common misconception about agile software development is that it requires no project planning. While in some respects it requires less project planning than projects developed using the traditional system development lifecycle approach, it does require planning. Further, and this is central to the point of this paper, the very nature of agile development requires the continual readjustment of the development plan. This is key to the contribution of the heuristic method described herein.

REFERENCES

[1] Ammann, P. and Offutt, J. *Introduction to Software Testing*, Cambridge University Press, New York, NY, USA, 2008.

[2] Artigues, C., Michelon, P., and Reusser, S. "Insertion Techniques for Static and Dynamic Resource-constrained Project Scheduling," *European Journal of Operational Research*, Volume 149, Number 2, 2003, pp. 249-267.

[3] Ashtiani, B., Leus, R., and Aryanezhad, M.-B. "New Competitive Results for the Stochastic Resource-constrained Project Scheduling Problem: Exploring the Benefits of Pre-processing," *Journal of Scheduling*, Volume 14, Number 2, 2011, pp. 157-171.

[4] Austin, R. D. and Devin, L. "Weighing the Benefits and Costs of Flexibility in Making Software: Toward a Contingency Theory of the Determinants of Development Process Design," *Information Systems Research*, Volume 20, Number 3, 2009, pp. 462-477.

[5] Bartels, J.-H. and Zimmermann, J. "Scheduling Tests in Automotive R&D Projects," *European Journal of Operational Research*, Volume 193, Number 3, 2009, pp. 805-819.

[6] Black, R. *Critical Testing Processes: Plan, Prepare, Perform, Perfect*, Addison-Wesley, Boston, MA, USA, 2003.

[7] Copeland, L. *A Practitioner's Guide to Software Test Design*, Artech House, Norwood, MA, USA, 2004.

[8] Craig, R. D. and Jaskiel, S. P. *Systematic Software Testing*, Artech House, Norwood, MA, USA, 2002.

[9] Crispin, L. and Gregory, J. *Agile Testing: A Practical Guide for Testers and Agile Teams*, Addison-Wesley, Boston, MA, USA, 2009.

[10] Dillon, R. L., Paté-Cornell, M. E., and Guikema, S. D. "Optimal Use of Budget Reserves to Minimize Technical and Management Failure Risks during Complex Project Development," *IEEE Transactions on Engineering Management*, Volume 52, Number 3, 2005, pp. 382-395.

[11] Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N. B. "A Decade of Agile Methodologies: Towards Explaining Agile Software Development," *Journal of Systems and Software*, Volume 85, Number 6, 2012, pp. 1213-1221.

[12] Gregory, J., and Crispin, L. *More Agile Testing: Learning Journeys for the Whole Team*, Addison-Wesley, Boston, MA, USA, 2014.

[13] Hanne, T. and Nickel, S. "A Multi-objective Evolutionary Algorithm for Scheduling and Inspection Planning in Software Development Projects," *European Journal of Operational Research*, Volume 167, Number 3, 2005, pp. 663-678.

[14] Hass, A. M. J. *Guide to Advanced Software Testing* (2nd ed.), Artech House, Norwood, MA, USA, 2014.

[15] IEEE Std. 829-2008. *IEEE Standard for Software and System Test Documentation*, IEEE Computer Society, New York, NY, USA, 2008.

[16] Jorgensen, P. C. *Software Testing: A Craftsman's Approach* (4th ed.), Auerbach, Boca Raton, FL, USA, 2013.

[17] Kolisch, R. and Hartmann, S. "Experimental Investigation of Heuristics for Resource-constrained Project Scheduling: An Update," *European Journal of Operational Research*, Volume 174, Number 1, 2006, pp. 23-37.

[18] Lambrechts, O., Demeulemeester, E., and Herroelen, W. "Proactive and Reactive Strategies for Resource-constrained Project Scheduling with Uncertain Resource Availabilities," *Journal of scheduling*, Volume 11, Number 2, 2008, pp. 121-136.

[19] Larman, C. *Agile & Iterative Development: A Manager's Guide*, Addison-Wesley, Boston, MA, USA, 2004.

[20] Martin, K. and Hoffman, B. "An Open Source Approach to Developing Software in a Small Organization," *IEEE Software*, Volume 24, Number 1, 2007, pp. 46-53.

[21] Martin, R. C. *Agile Software Development: Principles, Patterns, and Practices*, Prentice-Hall, Upper Saddle River, NJ, USA, 2002.

[22] McConnell, S. *Rapid Development: Taming Wild Software Schedule*, Microsoft Press, Redmond, WA, USA, 1996.

[23] Myers, G. J. *The Art of Software Testing*, John Wiley & Sons, Hoboken, NJ, USA, 1979.

[24] Naik, K. and Tripathy, P. *Software Testing and Quality Assurance: Theory and Practice*. John Wiley & Sons, Hoboken, NJ, USA, 2008.

[25] Pinedo, M. L. *Scheduling: Theory, Algorithms, and Systems* (4th ed.), Springer, New York, NY, USA, 2012.

[26] Russell, R. S. and Taylor, B. W. *Operations and Supply Chain Management* (8th ed.), John Wiley & Sons, Hoboken, NJ, USA, 2014.

[27] Shore, J. and Warden, S. *The Art of Agile Development*, O'Reilly Media, Sebastopol, CA, USA, 2008.

[28] Stevenson, W. J. *Operations Management* (12th ed.), McGraw-Hill Education, New York, NY, USA, 2015.

[29] Subramaniam, V. and Hunt, A. *Practices of an Agile Developer: Working in the Real World*, The Pragmatic Bookshelf, Raleigh, NC, USA, 2006.

[30] Tseng, L.-Y. and Chen, S.-C. "A Hybrid Metaheuristic for the Resource-constrained Project Scheduling Problem," *European Journal of Operational Research*, Volume 175, Number 2, 2006, pp. 707-721.

[31] Tseng, L.-Y. and Chen, S.-C. "Two-phase Genetic Local Search Algorithm for the Multimode Resource-constrained Project Scheduling Problem," *IEEE Transactions on Evolutionary Computation*, Volume 13, Number 4, 2009, pp. 848-857.

[32] Whittaker, J., Arbon, J., and Carollo, J. *How Google Tests Software*, Addison-Wesley, Boston, MA, USA, 2012.

AUTHOR BIOGRAPHIES

Mark L. Gillenson is Professor and formerly Department Chair of Business Information and Technology at the University of Memphis. He is also the Director of the Big Data and Analytics Research Cluster in the university's FedEx Institute of Technology. He is the author of several books on database management and numerous journal articles. His current research interests include advanced database systems and software testing.

Michael J. Racer is Associate Professor of Marketing and Supply Chain Management in the Fogelman College of Business and Economics of the University of Memphis. He received his BA in Mathematical Sciences from Rice University, and his M.S. and Ph.D. in Operations Research from the University of California at Berkeley. He is the Associate Director of eSOL, the Enterprise Simulation and Optimization Laboratory, and the Associate Director for Supply Chain in the Center for Biofuels Energy and Sustainable Technologies. He has been a member of INFORMS since 1990, and is currently the Presidents for INFORM-ed. He has published in the *European Journal of Operations Research*, *Management Science*, *Annals of Operations Research*, *Interfaces*, *Journal of Professional Issues in Engineering Education and Practice*, and *IIE Transactions*.

Xihui Zhang is an Associate Professor of Computer Information Systems in the College of Business of the University of North Alabama. He earned a Ph.D. in Business Administration with a concentration in Management Information Systems from the University of Memphis, 2009. His work has been published in the *Journal of Strategic Information Systems*, *Information & Management*, *Journal of Database Management*, *Journal of Organizational and End User Computing*, *Journal of Computer Information Systems*, *Journal of Information Systems Education*, and *Journal of Information Technology Management*, among others. He serves as the Managing Editor of *The Data Base for Advances in Information Systems*, and he also serves on the editorial review board for several academic journals, including the *Journal of*

Computer Information Systems, Journal of Information Systems Education, and *Journal of Information Technology Management*.

Ruby E. Booth is a doctoral student in Business Information and Technology at the University of Memphis. She teaches critical thinking and project management, with a focus on the use of data analytics to solve business problems. Her research interests are in the area of human factors in cybersecurity.

John P. Dugan is a principal at Coroutine, a software consultancy that specializes in new product development. He has two decades of experience designing and building software systems for companies ranging from the Fortune 500 to early-stage start-ups. Mr. Dugan has overseen Coroutine's agile practices for the last ten years and routinely helps clients adopt new strategies for managing their internal processes. He lives in Boulder, CO, with his wife and two sons.