Process Oriented Guided Inquiry Learning (POGIL) for Computer Science

Clifton Kussmaul
Muhlenberg College
2400 Chew St
Allentown, PA 18104-5586 USA
1-484-664-3352
kussmaul@muhlenberg.edu

ABSTRACT
This paper describes an ongoing project to develop activities for computer science (CS) using process oriented guided inquiry learning (POGIL). First, it reviews relevant background on effective learning and POGIL, compares POGIL to other forms of active learning, and describes the potential of POGIL for CS. Second, it describes a sample POGIL activity, including the structure and contents, student and facilitator actions during the activity, and how activities are designed. Third, it summarizes current progress and plans for a NSF TUES project to development POGIL materials for CS. Finally, it discusses student feedback and future directions.

Categories and Subject Descriptors
K.3.1 [Computers and Education]: Computer Uses in Education – Collaborative learning. K.3.2 [Computers and Education]: Computer and Information Science Education – Computer science education.

General Terms
Algorithms, Experimentation.

Keywords
active learning, algorithms, communication, data structures, experience report, inquiry learning, POGIL, software engineering, teams.

1. INTRODUCTION
To improve student learning, enthusiasm, and retention, especially in science, technology, engineering, and mathematics (STEM) areas, educators have developed a wide variety of approaches to engage students, enhance learning, and emphasize attitudes and skills in addition to knowledge (often rote).

This paper describes an ongoing project to develop a set of process oriented guided inquiry learning (POGIL) activities focused on computer science, including topics in data structures and algorithms (DS&A) and software engineering (SE). First, it reviews relevant background on effective learning and POGIL, compares POGIL to other forms of active learning, and describes the potential of POGIL for CS. Second, it describes a sample POGIL activity, including the structure and contents, student and facilitator actions during the activity, and how activities are designed. Third, it summarizes current progress and plans for a project to development POGIL materials for CS. Finally, it discusses student feedback and future directions.

SIGCSE'12, February 29-March 3, 2012, Raleigh,North Carolina, USA.

2. BACKGROUND

2.1 Active Learning in CS
Active learning is not new in CS education, and experiences have been reported in a variety of papers; a few are summarized below. Baldwin [3] described experiences, benefits, and pitfalls with discovery learning, which broadly refers to learning through self-teaching. McConnell [17] discussed active and collaborative learning (ACL) in CS, in a four part series that reviewed the benefits of ACL and describes a set of ACL activities, associated risks and ways of addressing them, characteristics of and techniques to promote effective groups, and activity design tips. Gonzalez [10] reported on CS1 sections where each session was roughly 1/3 discussion, 1/3 lecture, and 1/3 ACL. Students from ACL sections did significantly better in CS2 than peers from traditional sections. Beck and Chizhik [4] reported a CS1 course where students spent roughly half of class on ACL exercises. Students in the ACL section did significantly better than peers in a traditional section, and that effect was found for both genders and a variety of majors. Sowell and colleagues [21] [22] describe experiences with active learning in CS3, a software development course, and a systems software course, including sample exercises, lessons learned, and qualitative and quantitative results; in general, student grades and course evaluations improved.

Not surprisingly, CS faculty often explore ways to use technology to improve or supplement ACL (e.g. [2], [20]). Although there is an obvious risk of adding unneeded technology, there is also the potential to develop tools that could benefit other disciplines.

2.2 POGIL
POGIL is based on learning science (e.g. [24]), and shares characteristics with other forms of active, discovery, and inquiry-based learning (e.g. [8]). As described below, POGIL combines effective learning practices in ways that are synergistic.

In POGIL, teams of learners (typically 3-5) work on scripted inquiry activities and investigations designed to help them construct their own knowledge, often by modeling the original processes of discovery and research [18]. The teams follow processes with specific roles, steps, and reports that help students develop process skills and encourage individual responsibility and meta-cognition. The instructor serves as an active facilitator, not a

lecturer or passive observer; active facilitation is a key aspect of POGIL. POGIL activities and processes are designed to achieve specific learning objectives; typically an activity is designed to focus on 1-2 (disciplinary) concepts and 1-2 process skills. The POGIL Project (www.pogil.org) offers a set of workshops to help faculty learn about POGIL theory and practice, including how to facilitate, evaluate, and develop POGIL activities.

POGIL activities generally follow a *learning cycle* with 3 phases [1]. In *exploration*, students look for trends or patterns in data they have collected or that is provided, and generate and test hypotheses to help understand or explain the data. In *concept invention*, the trends, patterns, or hypotheses are used to define a new concept or term; importantly, students have constructed understanding before the concept is introduced. In *application*, the new concept is applied in other situations or contexts to help students generalize its meaning and applicability. Thus, the scripted activity provides information and asks questions to guide students through the learning cycle and help them develop process and learning skills. (Alternatively, some POGIL activities introduce a concept, and follow it with application and exploration.)

Designing effective POGIL activities can be time-consuming, but supporting resources are available [11]. Generally, the author first identifies learning objectives and the focus of the activity. For examples, teams could analyze data, derive equations, or explore the behavior of a physical system. Next, the author creates a sequence of *key questions* that guide teams through the inquiry process. Finally, the author identifies and develops supporting information, such as prerequisites, glossary of terms, references, handouts, and subsequent assignments or projects.

Activities involve three types of key questions. *Directed questions* have definite answers, are based on material available to students, and provide a foundation for later parts of the activity. *Convergent questions* may have multiple answers, and require teams to analyze and synthesize information to reach non-obvious conclusions. *Divergent questions* are open-ended, do not have right or wrong answers, and may lead teams and individual students in different directions.

POGIL has been developed and validated over the last 15 years. POGIL has been used extensively in chemistry, and also in materials science and engineering [7], and CS [6], although POGIL does not yet appear to be well known in CS. Multiple studies have found that POGIL significantly improves student performance, particularly for average and below-average students. (e.g. [9], [13], [16]). Table 1 below summarizes data from 3 different experiments (in chemistry courses). In each case it compares final grade distributions between a lecture-based version of the course format and a later POGIL-based version.

Table 1: Summary of Experimental Data

#	format	%A	%B	%C	%D,W,F	statistics
1	pre: lecture	19	33	26	22	$\chi2=40.9$
	post: POGIL	24	40	26	10	$\alpha<0.005$
2	pre: lecture	20	20	27	33	$\chi2=70.1$
	post: POGIL	29	35	24	12	$\alpha<0.01$
3	pre: lecture	12	19	16	53	$\chi2=19.1$
	post: POGIL	9	32	31	28	$\alpha<0.005$

3. POGIL for CS

POGIL has particular potential for CS education. Software development is largely a problem-solving activity since background information is usually available, and POGIL helps students to develop their problem-solving abilities. Teams are important to most IT organizations, and POGIL helps students to develop important team process skills. POGIL also encourages students to collaborate and learn from each other rather than focusing on an instructor. In chemistry and other disciplines, effective POGIL activities seem to be adopted and adapted by other instructors and at other institutions. Thus, POGIL may provide useful models for reusable materials in CS education, where such reuse appears to be problematic.

POGIL also presents some distinct challenges for CS. Currently, not many effective POGIL activities exist in CS; thus, faculty need to invest significant time and effort developing them. CS courses and curricula are more varied than in chemistry, and portions of the content change more rapidly, making it more difficult to adopt or adapt materials at other institutions.

3.1 Sample Activity – Queues & Stacks

This section describes activity *Queues & Stacks (Q&S)*, which introduces queues and stacks, to show what a typical activity looks like, and what the facilitator and students do during class. It also describes common variations and some of the underlying motivations. The full activity is available online [15].

At the start of Q&S, the facilitator spends a minute or two providing motivation and background, and then distributes the activity to the student teams. Teams can be formed in many ways, e.g. at random, by ability (strong students together, weaker students together), or by student choice. The following sections contain text from the activity (in italics), followed by elaboration and discussion. For brevity, some text has been simplified, and some sections have been omitted (see notes below).

3.1.1 Introduction

*As computer scientists, we study **data structures** – ways to organize & structure data. Implementing a data structure has pedagogical value, but as good (lazy?) computer scientists, we don't want to re-implement a familiar data structure for every situation. Instead, we want to design an **abstract data type (ADT)** that captures the key ideas of a data structure and can be used in many different situations.*

*This activity will give you some experience analyzing requirements and designing ADTs. In particular, you will investigate **queues** and **stacks**. Before you start, complete the form below to assign a role to each person.*

Q&S uses the following roles:

- *Manager*: keeps track of time and makes sure everyone contributes appropriately.
- *Recorder*: records all answers and questions, so team members and the facilitator have accurate notes.
- *Speaker*: speaks on behalf of the team to the facilitator, other teams, or the entire class.

These roles are typical, but POGIL activities might have a variety of roles; for example, an activity involving equipment might have a *Technician*. Roles help provide multiple perspectives and ensure that all team members contribute. Students rotate roles for each activity, but teams usually stay together for a set of activities.

3.1.2 Queues - Real World

Consider the following situations:

- *People standing in a checkout line at a store.*
- *Suitcases or packages on a conveyor belt.*
- *Phone callers waiting on hold.*
- *Documents waiting to be printed on a printer.*

1. (2 min) All of these situations have some common characteristics. Identify 2 other similar situations.

*2. (3 min) These are all examples of **queues**. Summarize their key common characteristics in 2-4 sentences.*

3. (2 min) List any variations or exceptions to these characteristics. Review status with the facilitator before continuing.

Each team begins to work through the questions. Each question has an estimated duration to help teams manage time. The facilitator circulates around the classroom to monitor progress, clarify questions, and help address problems; teams are not expected to complete activities unassisted. These three questions focus on the *exploration* phase of the learning cycle. Question 2 is *convergent* (most teams will have similar answers); when most teams have answered it, the facilitator might take a minute or two and ask teams to report their answers, particularly if there are differences between teams. Question 3 is *divergent* (answers may vary), and provides an opportunity for faster teams to consider new ideas while slower teams catch up.

3.1.3 Queues – Abstract Data Types

*Queues are often described as **First-In, First-Out (FIFO)** or **Last-In, Last-Out (LILO)**.*

*4. (3 min) Based on the key characteristics of queues that you identified above, **list 4-5 key operations** for a queue ADT, and **rank them** by importance (1=high, 5=low).*

*5. (5 min) Define a **method signature** for each queue operation (starting with the most important) including an appropriate name, input parameters, and return types. You may include constructors, although Java interfaces do not normally specify constructors.*

```
public interface Queue<T> {
   public Queue<T>(int maxSize);
}
```

6. (2 min) Compare your queue interface with the set of operations listed in your textbook, the Java API, or Wikipedia. Summarize any differences, insights, or questions.

These questions are *convergent*, and focus on the *concept invention* phase of the learning cycle. When most teams have finished question 4, the facilitator might have teams report or post answers, and discuss any differences or questions that arise. As teams work on question 5, the facilitator might help teams review constructors, interfaces, and generics, as needed. If only a few students need help, they may receive it from teammates; if several teams need help, the facilitator might stop the activity to review key concepts with the entire class, or even postpone the activity until a later date. Thus, by monitoring team progress on the activity, the facilitator can quickly identify and address problems.

The activity uses syntax from the Java programming language and the JUnit testing framework, but could be adapted readily to other languages and frameworks, or pseudo-code.

3.1.4 Queues – Unit Testing

*7. (5 min) Describe **unit tests** to verify the behavior of the above queue methods. Focus on the most important or error-prone methods. **Each person** should work on **a different method**. Do not write JUnit tests; instead, use the **Unit Test Worksheet** for each method. Here is an example:*

***Class:** Queue*
***Signature**: public Queue<T>(int maxSize);*

Pre	***Inputs***	***Expected Result***	***Post***
none	*maxSize <= 0*	*Exception*	
none	*maxSize > 0*	*Queue object*	*check size*

8. (3 min) As a team, review the set of tests you developed, and adjust as necessary. Summarize your insights and questions. If there is time, reflect on ways that your team could have developed the unit tests more efficiently. Review with the facilitator before continuing.

Question 7 is *convergent*, and focuses on the *application* phase of the learning cycle, since students must apply their new concepts (of queues) in a different content (unit testing). Question 8 is more *divergent* and encourages teams to reflect on what they have learned and how their team functioned. As students and teams work on these questions, the facilitator might help teams review unit test concepts. If time permits, the instructor might also encourage students to consult with students working on similar methods in other teams. Again, this enables the facilitator to quickly identify and respond to problems.

The next three sections of the activity (not shown here) take a similar approach to stacks. This helps students recognize that the process of analysis, design, and testing can be applied to many different situations. Queues are introduced before stacks since the latter are often less intuitive. However, since stacks are easier to implement, the activity considers array implementations of stacks before queues.

3.1.5 Stacks – Array Implementation

A stack ADT can be implemented in multiple ways; you will see several approaches. For now, we will consider how to implement a stack using an array, e.g.:

```
public interface Stack<T> {
  // field(s)
  private T [] data;

  // constructor – allocate data
  public Stack<T>(int size) {
    this.data = (T[]) new Object[size];
  }
  // methods
}
```

*9. (3 min) Assume we have an empty stack. **Show the contents of the data array after each operation**. If you use additional fields (e.g. to keep track of positions in the array), add a column for each one and show how its value changes.*

	private T [] data				*other fields*		
Operation	*[0]*	*[1]*	*[2]*	*[3]*			
a. Create new data structure.							
b. Add 'A'.							
c. Add 'B'.							
d. Remove value.							
e. Insert 'C'.							
f. Get current size.							
g. Remove value.							
h. Insert 'D'.							

*10. (3 min) For each stack method, write a complete English sentence to describe **how it could be implemented** using an array, and its **O() performance**. Hint: None of the methods should be O(N).*

11. (2 min) Summarize the pros & cons of implementing a stack with an array. Review with the facilitator before continuing.

In one sense, question 9 prompts students to *apply* their new understanding of stacks, but in another sense it prompts them to *explore* array implementation as part of a new learning cycle. (Such overlap is not uncommon.) Question 10 leads students towards *formation* of an implementation approach which would likely be *applied* in a subsequent programming assignment. Question 11 prompts students to reflect on their learning.

The final section of the activity (not shown here) considers array implementation of queues, generally following the approach for stacks. As before, this helps students learn that similar processes can be applied in different situations. The facilitator might need to help students recognize some of the differences between stacks and queues, and how to address them efficiently.

3.1.6 Next Steps

Usually, teams finish the entire POGIL activity during class, where the facilitator is available to monitor progress and provide assistance if necessary. In some cases, teams may need to complete an activity outside of class.

A POGIL activity is usually followed by assignments, assessments, or other tasks to help solidify and/or assess student learning. For the Q&S activity, this might include:

- A written report summarizing what the team did and learned in the activity. This might be drafted by the Recorder and reviewed by other students.
- An individual peer assessment to help identify and correct problems within a team.
- A short quiz at the start of the next class meeting.
- Unit tests for the stack and queue interfaces.
- Array implementations of a stack and/or queue.

4. Project Plan

This section summarizes current progress and plans for a project to development POGIL materials for CS, supported by the National Science Foundation (NSF) Transforming Undergraduate Education in STEM (TUES) program. This project leverages the successful POGIL approach, extends it to CS, and explores some new directions that could benefit POGIL in other fields. Specifically, this project is developing, refining, validating, and disseminating two sets of 8-10 POGIL activities for CS. Set A focuses on data structures & algorithms (DS&A), including queues & stacks, lists, hash tables, recursion, searching, and sorting. Set B focuses on software engineering (SE) including project scheduling (e.g. work breakdown structures, Gantt charts, critical path), estimation (e.g. story points), project budgeting and financial planning, and elements of the unified modeling language (UML).

These activities are packaged as a set of handouts containing background information and questions, including space to write answers. Each activity packet contains appropriate supporting materials, such as worksheets, code samples, references, and rubrics. Each packet also contains overview information for facilitators, such as learning objectives, prerequisites, directions to prepare the activity, notes for facilitating, a brief history of the activity, and planned or suggested improvements.

As the activities are developed, they are reviewed by chemistry faculty with extensive POGIL experience, and by CS faculty who have completed introductory POGIL workshops and plan to use POGIL in their classrooms. An assessment expert guides and support more detailed assessment plans and activities. As activities are refined and validated, they are disseminated to the broader CS education community through papers, conference presentations, workshops, and a website [15].

Thus, this project seeks to produce materials for adoption by other CS faculty, and thereby improve the quality of CS education, and particularly for average and below average students. The project also seeks to identify and foster a POGIL community within CS. Table 2 summarizes the project timeline.

Table 2: Project Timeline
(AY=academic year; PA = POGIL activity)

Timeframe	**PI & co-PIs**	**CS Collaborators**
2011 Summer	plan assessment create/refine PAs	POGIL training review PAs
2011-12 AY	pilot/refine PAs show: talks	review/pilot PAs evaluate (baseline)
2012 Summer	create/refine PAs	review/refine PAs
2012-13 AY	refine PAs push: workshops	evaluate (baseline/treatment)
2013 Summer	create/refine PAs	create/refine PAs
2013-14 AY	refine PAs push: talks/workshops	evaluate (treatment) pilot/refine PAs

4.1 Project Status

To date, the project has drafted, piloted, and revised the following POGIL activities [15]:

A. Data Structures & Algorithms
   1. Searching (linear & binary)
   2. Queues & Stacks
   3. Linked Lists
   4. Sorting (mergesort & quicksort)

B. Software Engineering
   1. Project Scheduling (PERT, critical path)
   2. UML Use Case & Activity Diagrams
   3. UML Data Diagrams & CRC Cards

C. Programming Constructs & Techniques
   1. Unit Testing (JUnit)
   2. Error Handling & Exceptions
   3. Reading & Writing Files
   4. Inheritance

D. Other
   1. Teams & Roles (introductory)
   2. Mind Mapping
   3. Sequence Analysis (bioinformatics)

The programming construct activities involve specific syntax for Java and JUnit, but could be translated for other programming languages. The other activities would require only minor changes for other languages.

4.2 Assessment Plan

Table 3 summarizes the project assessment plan, which will be revised and expanded during 2011-2012, with assistance from an assessment expert. SIR-II is the Student Instructional Report II™ [5], and TDS is the Team Diagnostic Survey [23].

Table 3: Project Assessment Plan

	Goal	Techniques(s)
Students	Improve student learning outcomes.	Average grades. Qualitative assessment of selected assignments.
	Improve student affective outcomes.	Current (e.g. SIR-II), existing (e.g. TDS), & custom instruments.
	Improve student recruiting & retention.	Course enrollments & major/minor counts.
PIs & Collaborators	Develop & refine PAs.	Quarterly activity reports, peer review, interviews.
	Improve faculty affective outcomes.	Reflection, interviews.
	Enhance PAs with technology (FOSS).	Quarterly activity reports, peer review, interviews.
	Raise awareness (talks/papers/workshops).	Quarterly activity reports, interviews.
Other	Foster community for POGIL in CS.	Talk/workshop attendance & evaluation forms.

5. CONCLUSIONS

POGIL is based on learning science and has a proven track record in other disciplines. There are a variety of materials to help faculty develop and improve POGIL materials (e.g. [11] [12] [18]). POGIL classrooms are very different from lecture-based classrooms, but POGIL shares characteristics with other forms of active, discovery, and inquiry-based learning, so that faculty familiar with such approaches should not have difficulty adapting to POGIL.

5.1 Student Feedback

During 2009-2010 the author developed POGIL and other active learning activities [14] for a graduate course (roughly 20 students) on genetic algorithms, neural networks, and fuzzy systems [19]. The activities were presented as slides which were later posted for students to review. This reduced paper use and made it easier to tweak activities during class by adding steps or providing more information, and to manage the activity's pace.

At the end of the course, students were asked to briefly describe their experiences. The responses included recurring themes. Students were initially uncertain about an unfamiliar teaching and learning style, but then realized that they had mastered concepts and acquired useful process skills. Students also had some difficulty understanding the instructor's spoken language; however, individual teams could discuss the activity in the language(s) they preferred. For example:

> *Group discussions & exchanging ideas with other groups is a better one. This makes us think about it in a better way. As you come to each group to talk, we can clarify our doubts by discussing it with you. And you encourage us to think more about the topic.*

5.2 Future Directions

Future directions for this work include developing and revising more POGIL activities, extended and adapting them for courses and faculty at other institutions (including high schools and community colleges), developing POGIL activities in other areas of computer science and related disciplines, and working to develop a community of faculty using POGIL in computer science. We also hope to explore the potential of tools, such as wikis (e.g. TikiWiki) and learning management systems (e.g. Moodle), to help scale activities to larger courses and a broader community. Effective use of such tools could help the facilitator to monitor team progress, and enable student teams to more readily summarize and compare their results and conclusions.

6. ACKNOWLEDGMENTS

Thanks to the National Science Foundation (TUES program) under grant DUE-1044679, and to US-India Educational Foundation (USIEF) for a Fulbright-Nehru teaching award that provided opportunities and other support for this work.

7. REFERENCES

[1] Abraham, M. R. 2005. Inquiry and the learning cycle approach. In: Pienta, N. J., Cooper, M. M., Greenbowe, T. J., eds. *Chemists' Guide to Effective Teaching*. Upper Saddle River, NJ: Prentice Hall; 41–52.

[2] Anderson, R., Anderson, R., Davis, K. M., Linnell, N., Prince, C., and Razmov, V. 2007. Supporting active learning and example based instruction with classroom technology. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM, New York, NY, 69–73. DOI=http://doi.acm.org/10.1145/1227310.1227338.

[3] Baldwin, D. 1996. Discovery learning in computer science. In: *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*. ACM, New York, NY, 222-226. DOI=http://dx.doi.org/10.1145/236462.236544.

[4] Beck, L. L., Chizhik, A. W. 2008. An experimental study of cooperative learning in CS1. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. ACM, New York, NY, 205–209. DOI=http://doi.acm.org/10.1145/1352135.1352208.

[5] Centra, John A. 2006. *The Student Instructional Report II™: Its Development, Uses and Supporting Research*. Educational Testing Service (ETS). Retrieved Nov 8, 2011 from http://www1.ets.org/Media/Products/SIR_II/pdf/53228_sirII_white_paper.pdf

[6] De Palma, A. 2005. Engaging students through the guided-inquiry cycle. In: *Proceedings of Redesigning Pedagogy: Research, Policy, Practice*. National Institute of Education, Nanyang Technological University, Singapore.

[7] Douglas, E. P., Chiu, C. 2009. Use of guided inquiry as an active learning technique in engineering. In: *Proceedings of the 2009 Research in Engineering Education Symposium*. Palm Cove, Queensland, Australia.

[8] Eberlein, T., Kampmeier, J., Minderhout, V., et al. 2008. Pedagogies of engagement in science. *Biochemistry and Molecular Biology Education*. 36(4):262-273.

[9] Farrell, J. J., Moog, R. S., Spencer, J. N. 1999. A guided-inquiry general chemistry course. *Journal of Chemical Education*. 76(4):570.

[10] Gonzalez, G. 2006. A systematic approach to active and cooperative learning in CS1 and its effects on CS2. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM, New York, NY, 133–137. DOI= http://doi.acm.org/10.1145/1121341.1121386.

[11] Hanson, D. M. 2005. Designing process-oriented guided-inquiry activities. In: Beyerlein, S. W., Apple, D. K., eds. *Faculty Guidebook - A Comprehensive Tool for Improving Faculty Performance*. 2nd ed. Pacific Crest.

[12] Hanson, D. M. 2006. *Instructor's Guide to Process-Oriented Guided-Inquiry Learning*. Lisle, IL: Pacific Crest.

[13] Hanson, D. M., Wolfskill, T. 2000. Process workshops - A new model for instruction. *Journal of Chemical Education*. 77(1):120.

[14] Kussmaul, C. 2011. Process oriented guided inquiry learning for soft computing. *International Conference on Advances in Computing & Communication (ACC)*, Kochi, Kerala, India.

[15] Kussmaul, C. 2011. *CS-POGIL: Process Oriented Guided Inquiry Learning in Computer Science*. Retrieved Nov 8, 2011 from http://cspogil.org.

[16] Lewis, S. E., Lewis, J. E. 2005. Departing from lectures: An evaluation of a peer-led guided inquiry alternative. *Journal of Chemical Education*. 82(1):135.

[17] McConnell, Jeffrey J. 2005. Active and cooperative learning: tips and tricks (part I). *ACM SIGCSE Bulletin* 37 (June): 27–30. DOI= http://doi.acm.org/10.1145/1083431.1083457.

[18] Moog, R. S., Spencer, J. N. eds. 2008. *Process-Oriented Guided Inquiry Learning (POGIL)*. Oxford University Press, USA.

[19] Munakata, T. 2008. *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More*. 2nd ed. Springer.

[20] Schweitzer, D., and Brown, W. 2007. Interactive visualization for the active learning classroom. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM, New York, NY, 208–212. DOI= http://doi.acm.org/10.1145/1227310.1227384

[21] Sowell, R., Chen, Y., Buhler, J., Goldman, S. A., Grimm, C., and Goldman, K. J.. 2010. Experiences with active learning in CS 3. *Journal of Computing Sciences in Colleges* 25 (May): 173–179.

[22] Sowell, R., Gill, C., Chamberlain, R. D., Grimm, C., Goldman, K. J., and Tranel, M. 2010. The active-learning transformation: a case study in software development and systems software courses. *Journal of Computing Sciences in Colleges* 25 (May): 165–172.

[23] Wageman, R. et al. 2005. Team Diagnostic Survey. *The Journal of Applied Behavioral Science*. 41, 4 (Dec. 2005), 373 -398.

[24] Zull, J. 2002. *The Art of Changing the Brain: Enriching the Practice of Teaching by Exploring the Biology of Learning*. 1st ed. Stylus Publishing.