120th ASEE Annual Conference & Exposition
FRANKLY, WE DO GIVE A D*MN
June 23-26, 2013

Paper ID #5766

An Educational Software Lifecycle Model Inspired by Human Physiology

Dr. Feras A. Batarseh, University of Central Florida

Feras Batarseh received the PhD degree in Computer Engineering (Software Engineering track) from the University of Central Florida (Orlando, FL, USA) in 2011. His research interests include the field of software engineering, and to date his focus has spanned the areas of software testing, validation and verification, artificial intelligence, knowledge-based systems and e-learning. He is a member of the ACM, ASEE and IEEE computer societies.

AN EDUCATIONAL SOFTWARE LIFECYCLE MODEL INSPIRED BY HUMAN PHYSIOLOGY

Abstract

Artificial Intelligence methods are frequently applied to projects of developing systems endowed with the intellectual processes in humans, such as the ability to reason, discover meaning, generalize, or learn from past experiences. However, the question remains, Can a man-made design/artifact be considered conscious? This paper aims to establish a direct relationship between the human physiology and Software Engineering, for educational purposes. Teaching Software Engineering can be challenging in cases when taught to non-engineering students. The class curriculum needs to be planned and structured to match the background of the students who are taking the class. Nowadays, students from majors other than Computer Engineering and Computer Science often are required to register for software-related classes, usually the introductory ones. Additionally, new fields are emerging between life sciences and software engineering, such as bioinformatics and computational chemistry, thus, an increasing need to address such fields. The model presented in this paper is called the Human Physiology Lifecycle Model for Learning (HPML). It is targeted towards students in the fields of biological, medical and life sciences (this includes biology, chemistry, medical studies, nursing, pharmacy, bioinformatics and public health majors). Students in these fields need to acquire the basics of engineering and software, medical software tools and a general understanding of computers.

Human physiology had proven itself to be a successful model to follow[1], or at least an inspirational one for science[4]. Especially in software engineering, fields such as genetic algorithms[3], computer vision[4], and computer scent recognition[2] are all examples on how to build software systems similar to biological systems; but could this be applied to education? To answer this question, this paper introduces a software model that follows the human physiology to structure different parts of a conventional software system; and to introduce it to students. On the other hand, lifecycle models can help in illustrating the different phases of software construction to students. HPML uses lifecycle models, for instance, HPML equates the human immune system to a safety and security software system (or module within a system); under this system students can learn about topics such as viruses, malware attacks, errors, defects and relate that to similar functions in the human body. Another example is the circulatory system; which controls the movement of blood which is pumped from the heart to all the parts of the body. Similar to that functionality is the data flow in a software system, and all its related topics (data latency, data mining, system throughput…etc). It is anticipated that this model establishes a cornerstone to a comprehensive educational lifecycle model that is fully inspired by human body systems and their parts.

KEY WORDS
Human Physiology, Lifecycle Model, Software Engineering, Life Sciences, Artificial Intelligence.

1. Introduction and Background

This section introduces a background on Software Engineering lifecycle models, Artificial Intelligence approaches, and related Computer Science educational models. Students in life-sciences (as well as many other majors) need to be equipped with knowledge about software in general, and that's due to multiple reasons: 1) at different points of their careers they will use software tools, 2) they will need to get up to speed some of the medical equipment they use, 3) model biological and chemical problems/solutions, and 4) have general computer/software knowledge.

1.1 Lifecycle Models:

Building software is not a random process, rather a well defined and structured procedure. The same is said about teaching those software development procedures. Teaching software is a challenging practice that requires planning and structured course design. One more level of difficulty is specially present when the audience (students) come from majors other than those related to engineering and computers. This paper targets students with life sciences background, and the class taught is based on an educational lifecycle named: Human Physiology Model for Learning (HPML). It is a software lifecycle model dedicated for teaching and consists of nine modules. However, before introducing HPML, it is important to present lifecycle models in general.
Lifecycle models can help in building software systems by controlling the software development process. According to Sommerville[5], the traditional Software Engineering lifecycle phases are[5]:

1. Requirements definition: At this stage, the user’s functional, non-functional and domain requirements are defined.
2. System and software design: The anticipated system is designed. It is usually represented in graphs, such as data flow diagrams and sequence diagrams.
3. Implementation and coding: This is the stage where the design is transferred to code and all system parts are implemented. In most cases, this stage consumes more time than any other stage.
4. Integration and system testing: The system modules are integrated in this step. Integration usually causes problems such as mismatches and miscommunication between the modules of the system. Therefore, complete system testing is required to overcome those difficulties.
5. Operational Maintenance: This phase takes place after the system is deployed and while the users are using the system. It includes follow-ups and dealing with user issues.

This lifecycle is also called the Waterfall Model.

Other example lifecycle methods include the *spiral model*, the *evolutionary development model*, the *reuse–oriented model*, the *incremental development model* and the *extreme programming model*[5].These software life-cycle models presented here are not the only ones, but they are the best known models and have been widely used[5]. There are many differences between these models. Using different models may lead to different results or might require a different set of resources and different budgets. Each of the models

mentioned above is used for a different set of systems. Therefore, it is tapering to use only one of these models as basis for teaching software engineering. The waterfall model is used in many Software Engineering courses as the de-facto, but this is not the case in a real-world software development process. To enable the students to working with software in the real world, they need to grasp the concepts in terms that they understand and can relate to. That is the aim of HPML.
HPML lies between the fields of Software Engineering and Artificial Intelligence (AI). AI has been regularly used for problem solving, software development and optimization, but not much for educational purposes. No educational lifecycle model, however, was defined based on AI and Software Engineering combined. This paper aims to establish that connection. In the next section, a brief background of AI is introduced.

1.2 Introduction to Artificial Intelligence Approaches:

According to the Webster's definition, intelligent systems are "systems that perceive their environment and take actions which maximize its chances of success."[6] In order to display such functionality, the system must be indeed *intelligent.* Over the past 50 years there have been many efforts towards achieving Artificial Intelligence in machines. These efforts have resulted in significant advances in that field. Computer *agents* are a type of intelligent system that can interact with humans in a realistic manner. They have been known to beat the world's best chess player and locate hostages in a military operation[2]. A computer agent is an autonomous or semi-autonomous entity that can emulate a human. It can be either physical such as a robot, or virtual such as an avatar[1]. Artificial Intelligence raises the question of whether machines can think and learn akin to humans. The ability to learn should be part of any system that has intelligence. Intelligent systems must be able to adapt to changes in their environment. AI disciplines that are inspired by the human physiology include[1]:

1. **Genetic Algorithms (GA)** is a method that finds a solution or an approximation to the solution for optimization and search problems. GA use biological techniques such as mutation, crossover and inheritance[3].
2. **Neural Networks** are a learning paradigm inspired by the human nervous system. In neural networks, information is processed by a set of interconnected nodes called neurons[1].
3. **Machine Learning** happens when the agent learns by exploring its surrounding and figuring what actions are the most rewarding[1].
4. **Natural Language Processing (NLP)** is a discipline that deals with linguistic interactions between humans and computers. It is an approach dedicated to improving the human-computer interaction. This approach is usually used for audio recognition[1].
5. **Computer Vision** is when the computer captures and analyzes images of the 3D world. This includes making the computer recognize objects in real-time[4].

These five approaches above are examples of derivations from human characteristics, other AI approaches (that are not strictly inspired by human's physiology) include:
Swarm Intelligence: A decentralized approach that forecasts problem prediction; it is inspired by animals (such as: birds flocking and ants looking for food). This approach

aims to lead to a global intelligent behavior through local communication between the objects (i.e. ants/birds)[2].
Knowledge-based systems (expert systems): are intelligent systems that reflect the knowledge of a proficient person. Knowledge-based systems are a specific kind of intelligent system that makes extensive use of knowledge. They use heuristic rather than algorithmic approaches for decision making[7].
Reinforcement learning: is part of machine learning, and how a machine ought to take decisions and actions based on continuous feedback on its previous actions (inspired by psychology)[5].

This paper aims to use AI for educational purposes (i.e. establish an educational process that is inspired by human physiology) The rest of this paper is structured as follows, next section introduces related work, section 2 presents the human physiology model (HPML) with all its parts, section 3 presents results from a survey conducted at the University of Central Florida (UCF) in Orlando, FL and section 4 presents future plans, and conclusions.

1.3 Related Work

In a modern Software Engineering class, it is no longer sufficient to simply teach Software Engineering students about code and generic software concepts. The field of Software Engineering is flexible, and the content of technology taught at school will have changed almost before students reach their first job. Researchers tried to address this issue through different methods[8, 9, 10, 11], such as project-driven courses and problem based learning. However, not many methods that address this issues are presented in literature, and not many target a certain category of students (as HPML aims to establish). Nonetheless, some commonplace approaches include:

Problem Based Learning (PBL)[8]:PBL is a technique for constructing and teaching courses driven by problems. The defined problems act as a incentive for student actions. The class is presented as a set of problems, and rather than conventional knowledge, problems drive the course. That motivates the idea of learning through a "staged sequence of problems" presented in a unified context along with associated learning material. The 4 phases of PBL are presented in Table 1, below. It consists of defining the problem, then learning through steps of solving it and presenting the findings and hypotheses[8].

Model-Driven Engineering (MDE)[9]:MDE was created and integrated into a Software Engineering course at Florida International University (FIU). Using MDE, class projects are based on ongoing research to develop the Communication Virtual Machine (CVM) technology. CVM is used to model and realize communication models created using the Communication Modeling Language (CML- A domain-specific modeling language) and is the is the main part of the MDE teaching system. CMV consists of models that lead the learning process, and students can control these models and modify their flow based on what they learnt in class[9].

Table 1: Four Phases of PBL [8]

	Activity	By Whom
Phase 1	• present problem • set hypothesis • identify learning • assign tasks	• tutor • group (with tutor) • group (with tutor) • group (with tutor)
Phase 2	• consult resources • detailed study of issues	• individual and group • individual
Phase 3	• evaluate resources • re-examine the problem • revise hypothesis and learning issues	• group (with tutor) • group (with tutor) • group (with tutor)
Phase 4	• present findings • metacognitive critique • reformulate further hypotheses	• group to peers and tutor • tutor and peers • group (with tutor)

Software through game design[10]: This model focuses on the "fun factor" of projects and teaching through gaming. In games, most of software topics could be taught, as well as other computer science disciplines such as Networks, or Human Computer Interaction[10]. This is illustrated in Figure 1.

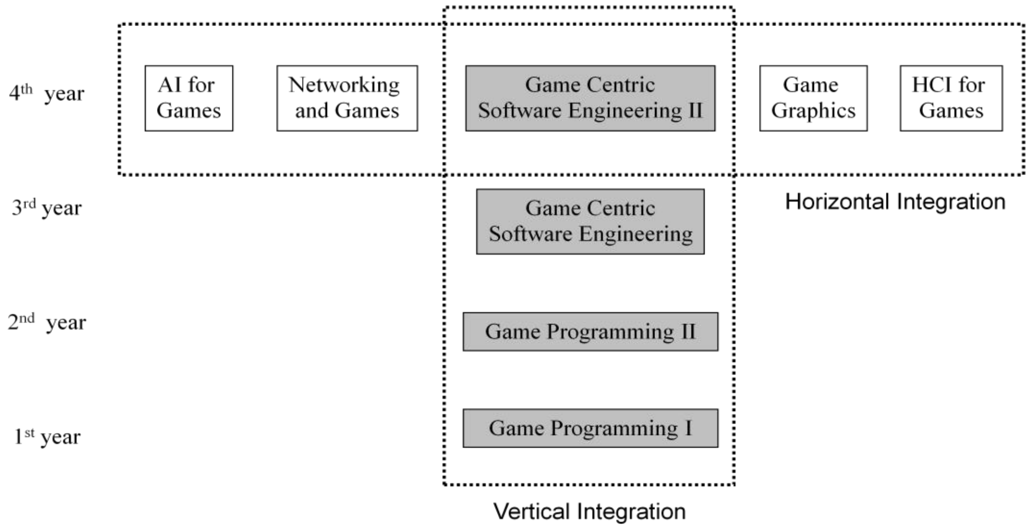

Figure 1: Software Game Design [10]

Practice Oriented Software Engineering[11]: As the name implies, this model is driven only by software projects practices. In software engineering, knowledge depends on practice and experience more than any other exercise. Authors claim that practice is the best way to help students when they graduate and start undergoing real practice of software engineering. Building the practice project goes through six stages (which are the six stages of the course), these stages are show in Figure 2 below.

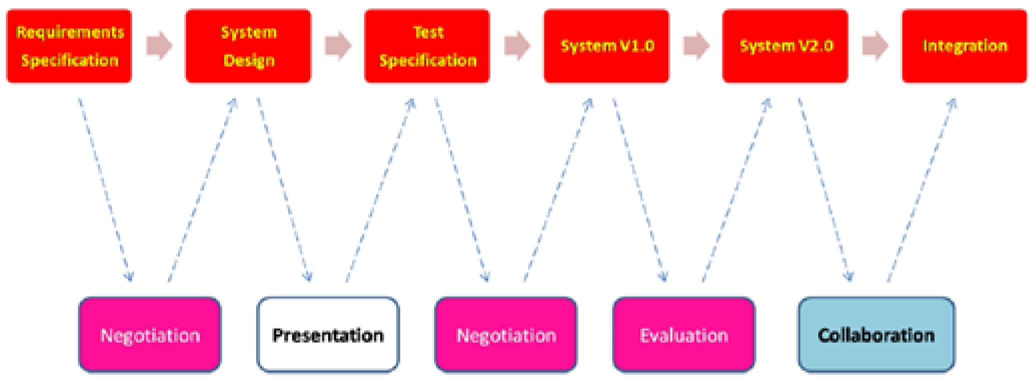

Figure 2: Practice Oriented Teaching [11]

As noted earlier, none of the existing models is specific to a certain category of students, and none provides a framework for students in life sciences. This paper aims to fill that gap, and provide a method that aids students in a specific category seize Software Engineering principles, and prepare them to utilize its concepts and principles in the real world whenever needed. The main contribution of this paper (HPML model) is presented in the following section. For every system, HPML covers a number of Software Engineering topics, these topics are listed and discussed next.

2. Human Physiology Lifecycle Model for Learning (HPML 1.0) (9 systems)

This paper introduces the first version of HPML (version 1.0). HPML could be used to structure classes such as: Introduction to Software Engineering, Principles of Software Engineering, Software Engineering Life Cycle Models, or Introduction to Computer Science. Although this model is aimed towards students in life-sciences majors, it could be also used with Computer Engineering and Computer Science students.
Modularity is a key characteristic in HPML. This section presents the modules inspired by human body systems and equates them with a software module. HPML consists of multiple systems (modules), each could serve as a stand-alone system, that is reusable and transferable to other projects. Integrating all these systems into one comprehensive system forms the complete body of the software system under study; each one of these nine systems could be introduced separately in class. The class shall be divided into nine chapters; tests also shall be structured into these nine modules. As part of HPML 1.0, in this paper, more emphasis is given to two very important body systems: Nervous and Digestive. The other seven systems are presented, however, these two systems drive the examples and analogies in this paper. The brain is presented next within the scope of the nervous system. Figure 3 illustrates all the human systems covered in HPML, and presented in this paper.

2.1 The Nervous System

One of the most important functions for the nervous system is decision making[12], which is centralized in the brain. In some cases, however, local decisions could be taken at the spinal cord for faster processing[12].

The brain is probably the most compelling part of human body, and the most challenging one to model[12]. Therefore, in HPML, the brain is introduced as a *business-oriented brain*, which manages the other bodily parts and works as the manager of the organization. While teaching the brain part in class, concepts such as Object Oriented (OO) design, principles of project management, communication between programmers and quality engineers, and data flow are introduced to the students. More about the brain is presented in the following section.

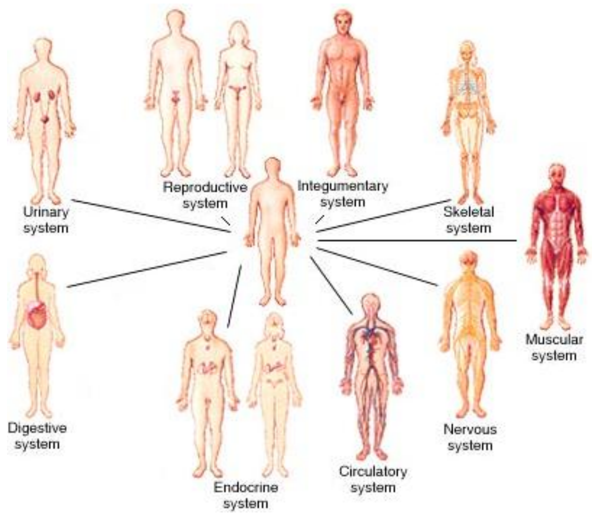

Figure 3: Human Physiology [12]

2.1.1 The Brain

The brain manages the body like an organization. HPML equates a software component (e.g., a Java OO class) with a corporate department or organization that provides internal services and/or products to other organizations within the company. The company then is equated to a complete software system.
Objects, or class instances, are identified with people and resources that work in those organizations represented by the classes. Methods correspond to job-related activities performed by employees that define the functional capabilities of the organization to which they belong. Message passing is equated to inter- and intra-organization communication conducted by employees interacting to accomplish organizational and corporate objectives.

Clearly efficiency and resource utilization are major concerns in both cases (organizations and software systems). This model aids managers to better understand the software system and help in the decision making process of what entities in the organization are consuming more/ less resources and in determining what are the most efficient entities by means of cost and time. Definition of the brain is based on the Object Oriented (OO) paradigm. HPML represents a common base between the two hierarchies (OO and business) in Table 2, and relate that to human body parts. The instructor in the class is recommended to present the brain functionality first and then discuss its equivalent in Software Engineering.

Table 2: Model Equivalences for the Brain Functions

Business entity	**Software system**	**Human body**
Corporation	OO system (usually a package)	The complete body
Organization	Subsystem (usually a class)	Body system (i.e. nervous system)
Management	Control method(constructor) or class(the main class)	The Brain
Employees	Class instances	Body parts (such as the hands and feet)
Work tasks(process)	Class methods	Human actions
Employees communication	Messages, data flow	Messages flowing from the brain to different body parts
Forms, memos, documents and files	Local data, methods parameters	Memory, knowledge, and past experiences

The measures of quality and effectiveness of the two types of systems is similar, or in better words related. The analogy of the system classes and the employees is a bit distorted, because people in organizations can do multiple tasks (of course the method can do multiple tasks, but a good design leads to a modular system and well defined tasks for each method in a class). However, in the context of human physiology, humans become sick (same as body parts), and the level of productivity changes from one day to another. In the case of software methods (assuming a valid system), it is either work fully or not work at all. As Table 2 indicates, both methods and employees need maintenance (encouragements, raises, promotions in the employees' case).
These similarities, differences and arguments establish good basis for education in class, and keeps the student involved. Students will continuously be looking for analogies and similarities and compare them with their field of study. Clearly the HPML analogy cannot be 100% precise; also those issues are not the core concern of HPML. HPML's main concern is education and the advancement of the student involvement in class.

An essential part of the Object Oriented paradigm is a class method. Methods are the main units of work, they could be classified whether they perform a service (update or delete) or produce a product (get the summation of two values and store them in a variable). Similar to the employees in an organization they either produce a product like a document or a plan, or they provide a service such as backing up the organization's data or make phone calls with customers. Figure 4 shows HPML's representation of the model structure of the OO software system, and Figure 5 shows a representation of a model structure from the business world.
Methods (presented by employees) have effect on their local data (departmental processes and decisions), or on the global data (organizational processes and decisions) of the class. Data is usually sent to methods with attributes (similar to messages in the business world). The secondary service provider is a software object that the primary service method might call to help perform its job (such as a personal assistance or a secretary in the business world). The relation between the amount of tasks performed by

the employee and his/her importance in the organization is directly proportional, same is the amount of calculations of statements executed by the method and its importance in the class.

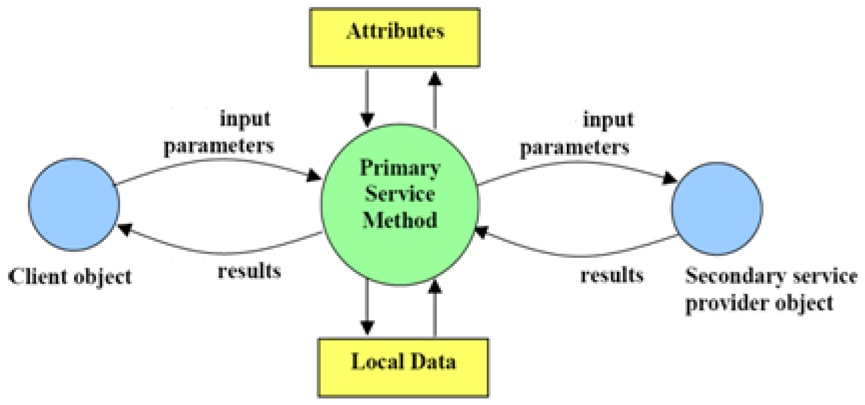

Figure 4: HPML Model for Object Oriented System

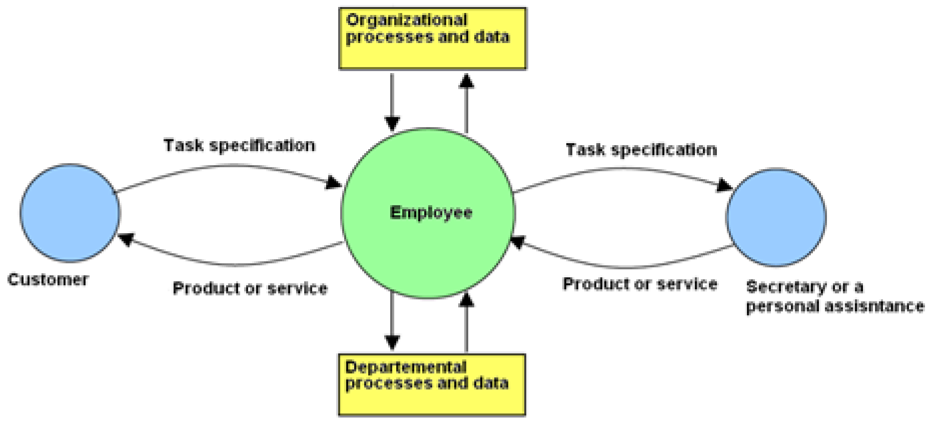
Organizational processes and data
Task specification
Task specification
Employee
Customer
Product or service
Product or service
Secretary or a personal assisntance
Departemental processes and data

Figure 5: Model from an Organizational Perspective

All these analogies aim to establish an understanding related to human physiology. Using HPML, by covering the nervous system in class, multiple software engineering topics are discussed and covered, these are:

1. Object Oriented Programming
2. Data Flow and Data Management
3. Data Mining
4. Project Management
5. Communication
6. Human factors in Software
7. Software Maintenance

Next section introduces the Digestive System, and focuses on the human tongue muscle as the entrance to that system. Parts of the Digestive System are equated to Software Engineering topics and presented to students.

2.2 The Digestive System:

The digestive system is the provider of energy in the human body[12]. In HPML, this part of the software system handles objects from users and other systems[12]. Other systems might interact to provide support, feed the system with required entities. In this paper, the tongue is the point of focus; it is the entry point and one of the most important parts of the digestive system. This human organ is mainly responsible for food intake and linguistic communications with the outside world. Other humans (being other systems in this educational model) are communicated with using certain protocols/languages and are distributed along different locations. Software systems need to be internationalized (i.e. speak multiple languages), therefore, the tongue in HPML represents the internationalization module in the software system.

2.2.1 The Tongue

In this section, an example is proposed, that could be used in class when presenting the tongue muscle[12]. Software Internationalization is presented as an example; it illustrates what could be presented/covered as part of one human body part. Students will be able to draw similarities between a multi-linguistic person and localization of software. The challenges of internationalizing a software is similar to that of a human learning a new language. These similarities include: learning the structure of the language, learning language's format, cultural differences, regional and date settings, a different currency, different orientations, protocols and differences with the mother tongue.

Tongue/Internationalization Educational Material:

Internationalization (i18n) is the process of adjusting software systems to enable support for multiple languages. In today's world, software vendors need to enter the global market to be able to expand and reach millions of users. So far, non-English language users have accepted software as it is (English/ASCII). However, this shortcoming is not acceptable anymore. Building multilingual software requires extra time and effort. Over the last years, internationalization has progressed into a must-do for many businesses. Any company/software vendor that wants to make their products global should invest manpower, time and money to reach global users. The process of modifying the software to include support for multiple languages, date formats, regional settings and cultural issues is referred to as software internationalization (i18n). Internationalization is based on localization (l10n), which is the adaptation and translation of a software system to different languages and a number of regional, date and currency settings. The word localization is derived from the technical term: locale[13, 14]. L10n is short for localization (10 letters between the letters *L* and *N*) and i18n is short for internationalization (18 letters between the letters *I* and *N*). Locale files are the basis on which localization is performed. It is important to note that the process of globalization is not a trivial process where strings are translated and hardcoded into the system, rather, the process is done by obtaining different files based on the settings of the machine, the desires of the user and applying the right locales[15,16]. Example projects and organizations that are specialized in software internationalization are: Li18nux (Linux internationalization initiative. It focuses on a core set of Linux APIs)[13, 14], The CITRUS Project (Comprehensive

Internationalization Framework towards Respectable UNIX systems. A project to implement locales for UNIX operating systems)[18], Mojikyo (a set of Asian double-byte languages) [19], and TRON (an internationalized architecture handled by T-Engine for embedded software systems)[20]. In the tongue module, different languages have different orientations and fall into different groups. Dealing with these groups is accomplished by using different *code-pages* and *Unicode fonts*. Languages fall into two main families, single-byte (such as: French, German, and Polish) and double-byte (such as: Japanese, Asian, Korean). Another Categorization of languages is based on their orientation. Most languages are Left-to-Right (LTR) (such as: English, Spanish) but some are Right-to-Left (RTL) (such as: Arabic and Hebrew)[13]. Furthermore, character encoding should be enabled for internationalization to perform correctly; it applies to the text, database and any other part of the system. The default encoding for a given system is determined by the runtime locale set on the machine's operating system. The most commonplace character encoding format is UTF (USC transformation format) USC is the Universal Character Set. UTF is designed to be compatible with ASCII. UTF has three types, UTF-8, UTF-16 and UTF-32. UTF is the international standard for ISO/IEC 10646 [15,16]. Lastly, multiple currencies (ex: Dollar: $, Euro: €, Yen: ¥) and date (MM/DD/YYY or DD/MM/YYYY) formats require support in the system[13,14].

Presenting the material in this example in class, shall be sufficient to cover the topic of internationalization in an introductory Software Engineering course. Using HPML, by covering the digestive system in class, multiple software engineering topics are discussed and covered, these are:

1. Software Communication Protocols
2. Software and Data Internationalization
3. Distributed Systems
4. ASCII, code pages, HTML and XML
5. Natural Language Processing

Next section presents the remaining seven systems of HPML.

2.3 Other HPML Human System

This section introduces seven modules of HPML covering a wide range of Software Engineering topics. These systems are:

1. **Musculoskeletal System**: Sub-system that's responsible for definitions and main skeleton of the system such as the classes definitions mainly the header files needed and libraries (this method already used in programming C++). By covering the Musculoskeletal System in class, the following topics are covered: System Design, System Planning and Software Libraries[12].
2. **Circulatory System**: In the human body the circulatory system is responsible for the movement of blood which is pumped from the heart, same as the data flow in the system. This subsystem (body part) is responsible for data flow between objects and classes. This subsystem is also the initiator of the system (equal to a Pacemaker in the human body- which is the part responsible for the first heart beat of the human being). By covering the

Circulatory System in class, the following topics are covered: Data Analysis, UML and other Systems and Data Flow Diagrams[12].

3. **Urinary System**: The Urinary system in the human physiology is responsible for removing the waste elements from the body, and so the same in the software systems. A similar example is the garbage collector in java, and destructors in object oriented programming languages. This sub-system is responsible for the destruction of un- wanted objects such as dangling pointers. By covering the Urinary System in class, the following topics covered: Software refactoring, Defect Detection, Software Testing, Validation and Verification[12].

4. **Immune System**: Similar to a safety and security sub-system. By covering the Immune System in class, the following topics are covered: Software Security, Computer Networks Security, Software Safety measures[12].

Table 3: HPML Modules

HPML Module	Software Engineering (Computer Science) Topics
The Nervous System	Object Oriented Programming, Data Flow, Data Management, Data Mining, Project Management, Communication, Human Factors in Software, and Software Maintenance
The Digestive System	Software Communication Protocols, Software Internationalization, Data Internationalization, Distributed Systems, ASCII, Code Pages, HTML & XML, and Natural Language Processing
The Musculoskeletal System	System Design, System Planning, and Software Libraries
The Circulatory System	Data Analysis, UML, and Data Flow Diagrams
The Urinary System	Software Refactoring, Validation and Verification, Defect Detection, and Software Testing
The Immune System	Software Security, Computer Networks, Security, and Software Safety measures
The Endocrine System	Data Communication, Web Protocols, and Network Communication
The Reproductive System	Inheritance, Sub-systems Design, and Software Integration
The Integumentary System	Computer Sensors, Computer Graphics, Computer Vision, and GUI Design

5. **Endocrine System**: The endocrine hormones serve as signals from one body system to another. In our proposed model this subsystem is responsible for data flow between objects, and messaging between system parts. By covering the Endocrine System in class, the following topics are covered: Data Communication, Web Protocols, and Network Communication[12].

6. **Reproductive System**: Sub system responsible for creating instances of objects, inheritance functionalities and relations between children, super classes and siblings similar to the human reproduction system (example: constructors creating instances of

objects). By covering the Reproductive System in class, the following topics are covered: Inheritance, Sub-systems design, Software Integration[12].
7. **Integumentary System**: In the human body, this system serves as a major sensory interface with the outside world just like the software's interface sub-system (OpenGL in C++ and Swing in Java are examples of interface classes). By covering the Integumentary System in class, the following topics are covered: Computer Sensors, Computer Graphics, Computer Vision, GUI design[12]. Table 3 presents all the HPML modules and their parts.

The next section presents the experimental survey, and the last section (4) presents conclusions and future plans for this research.

3. Experimental Survey

To collect feedback regarding the HPML approach, an informal experiment was conducted using surveys with 95 undergraduate students at the University of Central Florida. The students were asked if they prefer such an approach, and if they think that this is a better approach to learning Software Engineering. The results were grouped into 4 categories (shown in Figure 6) based on the students answers.
Category A: Students who think that the approach is good, but prefer to be conservative in their answer until they know more about HPML, and observe an actual course taught using its structure.
Category B: Students who valued the idea very much and hope to register for a class structured using HPML.
Category C: Students who disregarded the approach and thought its totally ineffective.
Category D: Students who think it's slightly effective but still prefer to learn using the classical approach.
The results of the survey are presented in Figure 6.

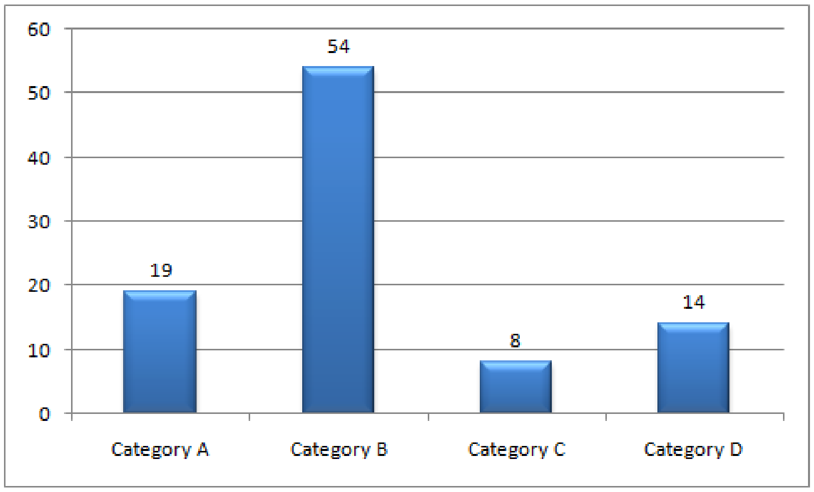

Figure 6: HPML Survey Results at UCF

Most students (54) provided very positive feedback, and only few of them (8) had very negative opinions. The rest of the students (Categories A and D) had neutral responses.

4. Conclusion and Future Plans

This paper presents a software life cycle model for software development (HPML). The digestive and nervous systems were presented in an extensive manner in this paper, other systems were also presented as part of HPML (ex: Circulatory, Immune and Reproductive). The main goal of this model is to teach Software Engineering for students in life sciences using structures and terms that they can relate to. Other students might benefit from this model too, particularly students who have focus between sciences and software, an example would be majors such as bioinformatics, where the student actually acquires knowledge from multiple fields. HPML was surveyed with 95 students at the University of Central Florida with successful results (54 students gave very positive feedback).

In this paper, nine systems were presented, an educational material example was discussed (software internationalization) and two major body parts were introduced in detail. All the systems (shown in Figure 3) were covered and all the modules were equated with Computer Science topics, presented in Table 3.
Future plans include using this model for teaching at the university, surveying it with more students at different educational institutions and improving it based on their feedback. Other plans include developing HPML 2.0 and providing a deeper insight into all the systems. HPML 2.0 will include defining the rest of the human physiology such as the Respiratory and Lymphatic systems.

Bibliography:

[1] Luger, G. F. “Artificial Intelligence: structures and strategies for complex problem solving” Fourth Edition, Published by Addison Wesley, 2002

[2] Book: Munakata, T., "Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More (Texts in Computer Science)", Second Edition, February 4, ISBN-10: 184628838X, 2008

[3] Book: Haupt, R., and Haupt, S. "Practical Genetic Algorithms", Second Edition, Wiley-InterScience, A John Wiley & Sons, INC., Publication, 2004

[4] Book: Shah, M., "Fundementals of Computer Vision", Published at the University of Central Florida, Orlando, FL, 1992

[5] Sommerville, I. “Software Engineering” 8th edition, Chapter 4, published by Addison Wesley, 2007

[6] The Merriam Webster Dictionary, Merriam Webster Incorporated 2009, http://www.merriam-webster.com/.

[7] Gonzalez, A.J. and Dankel, D. " The Engineering of Knowledge-Based Systems, Theory and Practice" Published by Prentice Hall 1993

[8] Barrows, H.S. "The Tutorial Process, Problem-Based Learning", Southern Illinois University School of Medicine, Springfield, IL., 1992

[9] Clarke, P. J., Wu, Y., Allen, A. A. "Experiences of Teaching Model Driven Engineering in a Software Design Course", International Conference of Model Driven Engineering Languages and Systems Educators Symposium, IEEE, 2009

[10] Claypool, K., and Claypool, M. "Teaching Software Engineering Through Game Design", ITiCSE 2729, Monte De Caparica, Portugal, 2005

[11] Coyne, R. F. et. al, "Teaching More Comprehensive Model-Based Software Engineering: Experience With Objectory’s Use Case Approach". 8th Conference on Software Engineering Education

(CSEE'95), New Orleans, LO, Lecture Notes in Computer Science, Linda Ibraham (Ed), Springer Verlag. April 1995

[12] Huang, W., "Anatomy and Physiology Series Introduction to Human Physiology Rapid Learning Center", Rapid Learning Inc. http://www.RapidLearningCenter.com

[13] Esselink, B. "A Practical Guide to Localization." John Benjamins Publishing Company, Amsterdam/Philadelphia. ANSI Z3948-1984 pp. 1-24, 2000

[14] Kresten, G. E., Kresten, A. A., and Rakowski, W. M., "Proceeding of the Journal of Global Information Management", Idea Group Publishing, pp.86-101, 2002

[15] Luong, T.V., Lok, J.S., Taylor, D.J. and Driscoll, K. "Internationalization: Developing Software For Global Markets." John Wiley & Sons, Inc. New York, USA, ISBN: 0-471-07661-9, 1995

[16] Aykin, D. "Usability and Internationalization of Information Technology." Lawrence Erlbaum Associates, Inc. New Jersey, USA, ISBN: 0-8058-4478-3, 2005

[17] The Linux Internationalization Initiative-li18nux (http://www.li18nux.org/)

[18] The Citrus Project (http://citrus.bsdclub.org/)

[19] The Mojikyo Project (http://www.mojikyo.org/)

[20] The TRON project (http://www.tron.org/index-e.html)