Injection of Business Coding Standards Practices to Embedded Software Courses

Donald V. MacKellar Jr.
Electrical and Computer Engineering Department
Gannon University
Erie, PA, USA
mackella001@gannon.edu

Abstract— **This paper details the process to inject and measure the understanding of student's knowledge and application of software coding standards. As with most other educational processes, consistent repetition enforces the learned concept. This process and techniques to reinforce the use of coding standards can be applied from freshman through graduate level embedded software courses.**

The process is set up around a living document outlining the requirements, expectations, and grading methodology. The standard is based on a compilation of industry standards and does not follow any particular standard explicitly. The intent is to educate the student as to "why" the industry uses such standards.

Evaluation of assignments to compliance of the standards, represent the business practice of code reviews. The measurable metrics enforces not only correctness of the assignment but compliance to a standard providing the skeletal framework of "how" to follow a standard.

To address why the embedded software industry follows these standards, we engage the student to live the process and experience the "why". Providing a course long assignment thread focuses the student's exposure to the "why". Early key assignments deal with standards and compliance, follow on assignments promoting code reuse.

Keywords— Business Injection, Industry Standards, Compliance Evaluation, EvalTools®

I. INTRODUCTION

Professional software engineers that work within industry must adhere to coding standards specific to the company, the specific industry in which they are employed, or to a governing certification organization demanded by an industry or government. There arises a need to educate and prepare students to understand the concept of following standards and processes utilized within professional settings[1].

Professional software engineers follow many standards, such as: Department of Defense (DoD) requires all suppliers follow DoD-STD-2167A [2], the Embedded industry in vehicle control systems formed the Motor Industry Software Reliability Association (MISRA) [3]. Service oriented companies provide suggested best practices for embedded software and processes [4]. The author's students were subjected to a similar concept through enforcement of standards where applicable, as defined in the author's *Programming Standards* [5] document developed for classes that develop software as a tool. The *Programming Standards* document defines the coding conventions, process, practices, and grading matrix that students should adhere to when developing and reviewing code.

The ISO standard for C and CPP programming languages allows for variations of implementation by the compilers and not all compilers meet the latest standards. The runtime behaviors of identical code compiled from various compilers are different. By teaching students to design code to a certain standard, the concepts of code reliability, maintainability and portability are promoted and the student is more ready for a configuration management oriented business environment. Implementation of the standard also introduces the student to the impacts standards can have to software performance metrics.

One study empirically proved that violating rules of standards, negatively impact software reliability [6]. This study used MISRA, an embedded system standard, focused on the prevention of faults by adopting coding rules as long as there is enforcement of the rules. This paper leverages the study of why standards exist and must be followed. This paper leverages the understanding of compliance of standards by using threaded assignments and enforcement of the standards. This way the student lives the process in lieu studying industry processes.

Language specifics are normally referred to as "coding styles". Many educational facilities suggest or refer to coding styles students should follow. The *Programming Standards* document is an attempt to introduce coding language styles as standards the student must follow leveraging a grading matrix and peer reviews.

Coding standards are an integral part of the business environment. The author intended to introduce this concept academically through enforcement and education via the *Programming Standards* document. The grader becomes the enforcer, education is provided through integrated lectures with properly defined examples using the standards set forth in the document, and assignments enhanced with the injection of the business process though "code peer reviews".

As a senior technical lead in embedded, real-time simulation systems, and as a professor the objective is to have the student standout in interviews and be aware of the expectations the hiring company will require to integrate within the business. The process surrounding the *Programming Standards* document is focused on evolving and improving the

student's ability to innovate within a set of externally defined constraints that are aimed to simulate a business environment.

The author tried to cover business practices, standards and processes without disrupting the course outcomes. The development and implementation of the *Programming Standards* document, the constancy of use throughout all of the author's classes, adherence to the standards in lecture and examples further reinforce the practice of compliance to standards. Thus preparing the new graduate to standout in interviews and better prepare them for the industry.

II. PROGRAMMING STANDARDS DOCUMENT

Professional software engineers follow "a standard <which>is a document that provides requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes and services are fit for their purpose" [7]. Depending on the industry and application there may be various penalties associated with not adhering to a given standard. Standards impact Business, Society, and Governmental regulations. The *Programming Standards* document is the method by which students in the author's classes are introduced to this concept and is designed around the following pillar: Any coursework or assignment is expected to be complaint with the *Programming Standards*. Penalties for non-compliance are communicated to the student thus setting clear expectations and responsibility for the grades earned. The intent is to educate and provide insight as to the importance of regulatory standards and the consequences of non-compliance.

The scope of the document describes the basic and measurable standards that require the student to develop software using the C or CPP languages. The intent is to have the student follow and learn basic software engineering processes. Not all elements in every homework or project assignment are addressed in this document. The student must hold to the deliverables provided in the requirements of the assignment.

If this document is referenced in the assignment, then the assignment must follow the standards. If there is a conflict between the assignment deliverable and this document, the assignment deliverable supersedes the document and the appropriate standard relevant to the conflict.

Each section provides information on why this is a required standard with any industry or business inferences. Then the section is further decomposed to the elements of the standard.

The layout of the documents is the assignment evaluation process followed by the details of the standards to various sections of the evaluation process.

A. Section 2: Grading

The first section provides the grading process. The student is presented the process and actions that will evaluate their work from submission in EvalTools®, the online assessment toolset used at Gannon [10] through execution or demonstration.

The intent of following a coding standard and process is to get the student familiar with how the software development industry does business, by following a process and standards. This will be conveyed to the student is though a strict policy of grading software submissions.

The term plagiarism in software is hard to distinguish. However, through the use of this standard specifically in sections dealing with comments and identifiers, it does become evident and easier to distinguish between group problem design sharing (which is learning) and code copying. Plus the use of exams and peer code reviews will highlight students not attempting to learn.

1) Grader's first action should be the student's last

This subsection describes the major steps and criteria in evaluating their assignment. This provides a standard process of not only *How* the process works but a standard for a grading assistant.

2) Evaluating the Submission to the Requirements and Engineering Process

This subsection describes the engineering process evaluation. After verifying the code compiles and links, the graders IDE is instrumented to verify the students' submission to the requirements of the assignment and standards.

This is a simplified software engineering Waterfall process traceability: Requirements Analysis (the assignment) to Design (header comment Scope), from Design to Implementation (Code) to Verification (developers test plan in header comment) to Validation (Grader's test plan).

3) Outside the Assignment Box or Too much Initiative

This subsection describes that adding more than what is required can promote undesirable software development behavior. Sometimes professors (and Managers) promote students and engineers to be enthusiastic and take initiatives. This is where the two provide additional thoughtful or insightful work in the assignment. This is great, but not for promoting process and standards in the classroom. Not only does it make the assignment evaluation more difficult to grade, but it could induce errors because the material was not covered or covered completely. In the business world, providing more than what is required sometimes shows the manager insightful vision, but it also wastes the time and effort for compensation where the effort could be applied to other challenges the manager needs completed.

4) Hail Mary

This subsection describes to the student the last ditch submission approach and consequences. Some students do not do their own work and many times they submit in-class examples (even last year's homework or someone else who is not in the class) in hopes that the code is just graded and not really reviewed. This is a case for aggressive grading, by submitting the author's or trying to snowball the grader.

5) Run-Time Errors/Traps/Exceptions

The final subsection deals with the implementation evaluation and requirements assessment process. During the execution phase of the grading process the grader has a validation test plan and monitors for expected results. Results

that yield wrong answers are normally logic errors related to the requirements. Sometimes assignments require exception handling; however, this section addresses unexpected execution exceptions.

Traps and unwarranted exceptions indicate that the student did not execute the code or follow any test plan. An exception is an interrupt which is unrelated to the student's program. This can be recoverable at times and are considered resume-able exceptions: floating point errors, stack abuse, mathematical errors, memory violations, etc. The term trap is used interchangeably with exceptions.

However, the student's assignment does not execute because the debugger caught the exception or the system crashed (stopped executing). Both mean that the code is broke and not tested completely by the student.

B. Section 3: Submission

This section is standard is for what files are to be uploaded to EvalTools® for homework and project submissions. The filename constructs which provide a homogenous look when grading. Deviations in the submission process induce additional time to setup the compiling and linking. This section describes the single, multiple file submission process, and team submissions.

Industry file naming conventions standard such as ISO 9660 are specific to a media or function. Most industry and certification standards do not define file naming conventions. Internal business standards focus on their product maintainability of large projects and do have file-naming conventions

C. Section 4: Source code format

This section provides an overall layout of the code. The reasoning for a format standard is during the internal business process, code reviews require side-by-side code comparisons. Adhering to a format standard, comparisons can be easily performed. Academically, this also provide less distractions to the grader when all of the source code submissions are formatted the same, evaluation of key requirements can be identified more easily. There are five major features to the source file: Header Comment field, Compiler Directives/Preprocessor Fields, Global Identifier Field, Code field, and inline Code comment standards.

D. Section 5: Comment Standards

This section provides the heart of the software design process and life-cycle as a measureable metric. The requirement analysis to design is captured in the header comment field as does the notion of configuration management and verification test plan. This section also provides the code layout style a standard for readability, understandability, and maintainability. To assess the student's knowledge of requirements analysis though design the use and correctness of comments are key. This dovetails to industry best practices and is a standard in most internal business. Code reviews are used as internal business process to enforce maintainability and conformance.

There are tools used to capture modified code and references which promote doing this activity automatically [12]. These tools are used when multiple engineers change large amount of codes for a release for code reviews. Other uses for differencing tools are when proto-type code is use for exploratory or analysis for control systems. However, it is the intent of the author to teach the behavior of designed change.

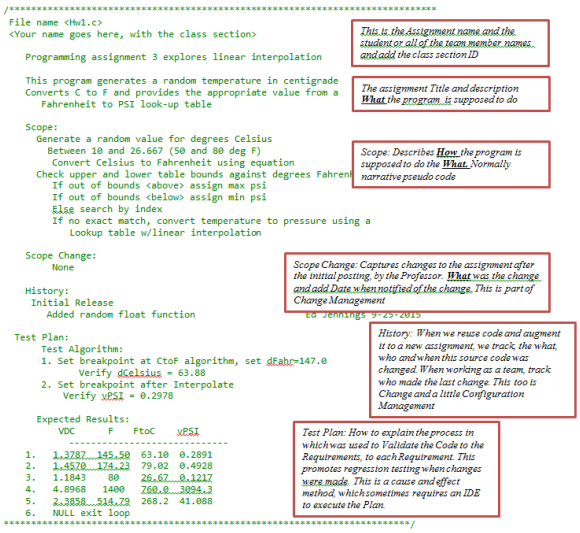

Figure 1: Example of Header Comment Field

The header comment field is the most important comment area of any assignment shown in Figure 1 [5]. It replaces the need for additional reports. It includes the purpose, design, revision and test plans in which the code was developed. However, if the code does not match the design in the header field, the grader is to assume the software process was not followed or some or all of the code/design was lifted from another source. The header comment field must contain the following:

Each source code function() requires comment field and the format it follows is similar to standards the author followed in current and previous occupations. These comments that describe the intent of the function(), the arguments in to the function and the return values developed. Examples of expected format and features are included in the standards document. The same grading criteria are followed for the function as for the header comment.

Finally inline comments are addressed as to function and style. These are normally specific to the source code language and are more subjective to evaluation. As a standard internal to the business, this promotes intent of the implementation to the requirement. And during the code review, knowledge transfer to the rest of the team.

E. Section 6: Identifier Naming Standards

This section provides the reasons and format of identifier notation. The standard defines how all identifiers (variables, constants, functions), macros and complex/user defined data types shall be notated. There are a few standards used in the industry as standard naming of identifiers: Positional Notation,

Composite Schema, Hungarian Notation, CamelCase, Multiple-Word, etc.

The identifier standard defined in the *Programming Standards* document is a variation of CamelCase notation. CamelCase is composed of a number of words(symbolic qualifiers) without white spacing where each word's initial letter is capitalized. CamelCase is a naming convention suggested by Sun Microsystems, Java communities, and other object oriented communities.

The variation is focused on value identifiers in which the compiler and linker defined as memory based values and function identifiers.

- Value Identifier : (symbolicQualifier(Units)(_state)
- Function Identifier : (SymbolicQualifier())

Examples and suggested Units and _state conventions are listed in the appendices of the standards document.

F. Sections 7 and 8:

These sections provide the reasons and format of indents, scoping alignment, and white spacing which focus on the readability of the student's source code. During peer reviews if these standards are not met, the meeting is terminated until the presenter has followed the basic internal business standards. This is analogous to the process and formatting rules of this paper's acceptance to FIE.

G. Sections 9 through 13:

These sections focus on the coding style and best practices which are considered standards. The use of numeric constants, casting, identifier declarations, use-case of parentheses, and execution path constructs "dangling else" are subject to the internal business standards. For student evaluation they are considered poor practices[4] and are to be avoided. These sections will expand as this pedagogical process continues. During peer reviews if these standards are not met, the meeting is terminated until the presenter has followed the basic internal business standards.

H. Section 14: Professional Relevancy

In the introduction of the standards document the reasons are defined for the inclusion of standards and business practices to the ECE courses the author teaches. However, the professionalisms between software engineers and their peers are not normally recorded in standards. Some are defined in 'coding styles" or "best practices" or in some cases assumed standards. Senior leads enforce these types of inherent standards during formal and informal (peer) code reviews. The undocumented or lightly documented standards are subjective standards and they define the attitude, professional, and character of the team.

In this section, various "styles and best practices" are addressed. To remove the subjective nature, these styles will be defined and become a measured outcome of Professional Relevancy to the student, when appropriate. To summarize this section, one could call this the "Sloppy Metric", the riddance of hackers, code writers (not software developers), arrogant developers, lazy software developers, just in time submission students, or just plain sloppy work. The subsections are listed below:

- Irrelevant Code aka Dead Code
- Unreachable Code
- Debug Code and Comments
- Irrelevant Comments
- Readability

I. Section 15: Business Practices: Reviews

Many standards and practices are employed throughout all industries and disciplines, but software follows some basic concepts. Code reviews are sometimes referred to Peer Reviews basically provide formal and informal processes to make sure the software development is following standards set forth by the business, certification agency, and/or industry. These reviews could be informal between developers or extremely formal technical reviews with senior and principle engineers. This section describes; definition of a code review, why they are important, and the goals [10].

III. INJECTION: TRAINING THE STUDENT

Two approaches were applied during the semester. The challenge was how to train freshmen and how to train rising seniors and graduates. The two courses being evaluated were ECE111 Introduction to C Programming registered with second semester freshmen and ECE347/GECE547 Embedded System Design, a cross-listed course between rising seniors and graduate students.

The approach for the freshmen was to introduce the standards as an integrated part of the course material and provide a consistent repetition of the standards through each topic. For example, when discussing "*C data types"* the section on identifier naming notation was covered. The assignment evaluation included proper naming conventions as per the standard. As each new course topic was started, there was discussion and references to the *Programming Standards* document. The evaluation was based on the following the standard of naming convention, not the correctness. Proper and relevant naming would be measured in the Key Assignments [11].

This approach of using Key Assignments, regressively trained the student through the course. This was a thread of assignments, where the beginning of the thread started with data typing and an algorithm. As new topics were added to subsequent Key Assignments, the student had to reuse their previously graded assignment. This promoted the understanding of code reuse, enforced many of the standards being followed such as readability, proper naming convention, and valid commenting practices.

On one assignment in this thread, the author anonymously swapped student's code. The more advanced students got the lesser advanced students and visa versa. This approach was to get the lesser advanced students a view of good practices and to enhance that code for new requirements. The more advanced

students had to correct or replace code. All code replaced or enhanced was to be commented out with a documented reason. The students had the opportunity learn and observe why businesses follow standards through this exercise.

This threaded assignment process also prepared the student to understand design for maintainability and introduced to configuration management. This thread ended with data structures and message mapping. Where that last assignment had a complete snap-shot of the semester.

For the other sections in the ECE347/GECE547 the students have had all of the basic programming classes. This means the injection cannot dovetail with the lectures and topics. The approach was if the students were newly hired engineers and they need to implement the standards immediately. This was not an unreasonable approach; it is that same injection they would face as if entering the business environment. A lecture was introduced on the standards and the scoring matrix. The first assignment grading cycle was harsh and by the book. The following lecture after grading, the author performed a code review on one of the student's homework submission using the peer review rubric the author developed. The rubric will be discussed in the next section. This approach demonstrated to the student the relevance of the standards and the author was able to articulate the reasoning of each criterion. This approach was slightly obtrusive to the lecture series and the extra time required was made up on a few longer class periods.

IV. ASSESSING THE STUDENT

There are two major assessments to address. One is the evaluation meeting the requirements of the assignment and the correctness by evaluating compliance to the coding standards. This emulates the standards in the industry or internal business practices. The other is the evaluation of process and understanding of the standards. This emulates the code review process, a process found in most industry standards for software development.

Each class and every Key Assignment requiring code development or enhancement had a requirement to follow the *Programming Standards* document. This requirement of the assignment informs the student that their deliverable(s) will be evaluated for not only providing the right answers of the assignment, but for correctness of the process. By integrating the standards in to the established Key Assignments as a deliverable, the student is scored to the penalty matrix.

A penalty scoring matrix shown in Figure 2 [5], is also included in the *Programming Standards* document, identifies the assessment criteria in the Key Assignments. The scoring matrix represents the max penalty score for each associated reference. The matrix provides an upper level objective score for the grader. For the student it provides a snapshot or checklist to make sure not only is the code right, but that the code is also correct and process has been followed to a standard. For reference, a standard Key Assignment has a forty-point possibility.

Section Reference	Standard's Scoring Penalties	Minus 1	Minus 3	Minus 5	Minus 10	Auto Score
3	Submission			X		
2.4	Hail Mary					0
2.1.3	Late Submission to EvalTools®			X		
Obvious	No Submission to EvalTools®					1
2.1	Fails Compilation / Linking					%50
2.5	Runtime Exception / Trap			X		
2.2.2	Fails Scope (design)/Code(implementation) ea.		X			
2.2.1	Fails Requirements/Scope(design)			X		
5.1	Header Comment				X	
5.2	Function Comment		X			
6.1	Identifier Notation	X				
5.3	In-line Comment	X				
6.2	Macro Notation	X				
7	White Space & Alignment ea. Max=5	X				
8	Scope Control	X				
9	Identifier Declarations ea.		X			
11	Use of Constants ea.		X			
12	(Cast) Commenting		X			
13.1.1	Dangling Else		X			
14	Professional Relevancy			X		

Figure 2: Penalty Scoring Matrix

Each course has an outcome that addresses the understanding of software practices using industry standards and review process. Depending on the course level, the course outcomes may differ in wording to meet the author's department Bloom taxonomy on course outcomes [11].

From the industry or business process, a code review is a systematic examination of computer source code. It is intended verify the design of the code meets the requirements, to find and fix mistakes overlooked in the initial development phase, improving overall quality and maintainability of the software, and help in the skills development and knowledge domain of the product by the team.

Academically, instead of performing code reviews or using various best practice or standard checklists, the students evaluate known code. This process is evaluating the student to a process: assessing the assessor. The code has already been evaluated by the instructor using a rubric from Figure 3. The students use the same rubric to evaluate the code. This rubric was setup to emulate basic common code review criteria with a twist. The rubric is tightly coupled to the *Programming Standards* document and is a guide to understand the principles of the standards. The criterion of the rubric follows the seven logical sections of the standard. Each descriptor of the rubric is the expected level of compliance to the standard with references to the appropriate sections in the standard.

The difference between the scoring matrix and the rubric, the matrix is used to verify correctness to the given assignment, the rubric to verify the compliance and completeness to the standards.

Each rubric criterion has an EAMU [11] performance vector which classifies student learning performance into four categories: Excellent (E), Adequate (A), Minimal (M), and Unsatisfactory (U). The rubric also provides a business performance mapping to EAMU: E: Compliant to Standards/Practice (9-points), A: Meets Intent: Code may require change(6-points), M: Inconsistent: Requires rework (3-points), and U: Incompliant: Requires retraining (1-point).

Name:	Student Id:	Class#/Section:	Semester/Year		
Criteria	**E: Compliant to Standards/Prac**	**A: Meets Intent: Code may require**	**M: Inconsistent: Requires rew**	**U: Incompliant: Requires retrain**	**Score**
Submission Process	The Submission of the code meets all of Section 3 of the coding standard where applicable.	The Submission of the code meets Sections 3.1.1, 3.1.2, and 3.1.3 of the coding standard and missed section 3.2.1.	The Submission of the code meets at least Sections 3.1.1 and 3.1.2 of the coding standard.	The Submission of the code did not meet Sections 3.1.1 or 3.1.2 of the coding standard.	
Comments / Documentation	The code meets the Content and Format of Section 5.1, 5.2, and 5.3. Where 5.3 describes most logic paths and code style sections. Use Appendix B and C for reference to format and content of section 5.1 and 5.2	The code meets **most** of Content and Format of Section 5.1, 5.2, and 5.3. Where 5.3 describes most **some** logic paths and **some** code style sections. Use Appendix B and C for reference to format and content of section 5.1 and 5.2	The code **less than 50%** Content **or** Format of Section 5.1, 5.2, and 5.3. Where 5.3 is **non-existent.** Use Appendix B and C for reference to format and content of section 5.1 and 5.2	There is only a trace (less than 10%) or no attempt to meet Section 5.1, 5.2, and 5.3. Where 5.3 has only trace or no conformance.	
Naming Notations	All Value and Function Identifiers Notation follow Section 6.1 Camel Case conventions. Examples are in 6.1.x and Appendices D and E. All Identifiers have relevant names to the Requirements or Usage. Single letter Identifiers are prohibited.	All Value and Function Identifiers Notation follow Section 6.1 Camel Case conventions. Examples are in 6.1.x and Appendices D and E. **Most** identifiers have relevant names to the Requirements or Usage. **Some** single letter identifiers are defined.	**Most** Value and Function Identifiers Notation follow Section 6.1 Camel Case conventions. **Most** identifiers have relevant names to the Requirements or Usage. Usage of single character identifiers is common and **are not** commented	No nameing convention is followed or the wrong one is followed. Usage of single character identifiers is common and **are not** commented	
Code Style	Code style is very dependent on the code language. However, basic standards must be met. Sections 8 thru 13 pertain to strict adherence to coding independence of the language. The code must meet each standard.	The code style misses the lesser or lower impact standards in Section 8 and 12 **or** meets 75% of Sections 8 thru 13.	The code style misses the lesser or lower impact standards in Section 8, 9, 12 **or** meets 50% of Sections 8 thru 13.	The code style violates standards in Section 8, 9, 11, 12 **and** 13.	
Compilation and Linking	First action to grading is compilation and linking. Section 2.1.1 must pass per the submitted code.	The Code is not in a compile-able textual format but can be, if simply changed by the grader to a known language construct. In other words, the submission is not in an approved file extension, Section 2.1.2. For example: .doc or .txt files, by removing the cover page and changing the file extension notation the submission	Code compiles but does not link. This could be the result of non-submitted support files or wrong header declarations, etc.	Code does not compile due to syntax errors.	
Execution / Runtime	Code standard 2.5 run-time errors, traps, or unexpected exceptions **cannot** occur during the evaluation run of the executable code.	IDE / system level warnings occur during the execution of the code. These conditions do not terminate the code execution. This could indicate that the student may have used a different IDE version or applied more conservative options to the IDE project preferences.	Floating point exceptions may occur, which terminate the code's execution. This may be caused by the student not performing a properly designed test plan. The grader is following a designed test plan to expose requirements conformance and code design knowledge domain. This is inadequate test plan design or rush to submission of assignment.	Exceptions other than FPE. occur due to inadequate or no testing at all.	
Meeting Requirements	Correct(process met and right answers). Each Requirement is met in the design documentation (scope field of header comment) and applied (code logic) meets the grader's test plan.	Right Answer (**all** of the test plan passes), **most** of the design is correct(scope field of header comment meets **most** of the requirements): The code execution meets **all** of the grader's test plan (the expected answers) and **most** requirements are captured in the design documentation (scope field of header comment). The code execution (code logic) is inconsistent to the design documentation.	**Mostly** Right Answers (**most** of the test plan passes), **most** of the design is correct(scope field of header comment meets **most** of the requirements): The code execution meets **most** the grader's test plan (the expected answers) and **most** of the requirements are captured in the design documentation (scope field of header comment). The code execution (code logic) **and** the design documentation **meet most** of the requirements.	**Some/No** Right Answers (**some or none** of the test plan passes), **Some/None** of the design is correct(scope field of header comment meets **some or none** of the requirements): The code execution meets **some or none** the grader's test plan (the expected answers) and **some or none** of the requirements are captured in the design documentation (scope field of header comment). The code execution (code logic) **and** the design	
Professional Relevancy	Professional Relevancy: Section 14: measures the soft skills also known as the styles and best practices not normally documented as an industry standard, yet employed throughout as professional software developers. No dead code, no unreachable code, no debug code/comments, no irrelevant comments, **and** the source code meets **all** readability	Section 14: No dead code, no unreachable code, no debug code/comments, no irrelevant comments, **and fails** to meet **any** of the readability Sections 4, 7, and 8.	Section 14: No dead code **and** no irrelevant comments **and fails** to meet **any** of the readability Sections 4, 7, and 8. Debug code and comments remain. Proof that the code might be owned by the student/developer	If the source code cannot meet the M: Inconsistent metric. The student cannot articulate few if not any portions of the code. This puts the code under suspect of plagiarism.	

Figure 3: Peer Review Rubric

V. Lessons Learned and Conclusion

This injection of business coding standards and practices was applied to all software-oriented classes the author taught during a semester. Each class had some form of a consistent repetition to help enforce the concepts of following standards. The courses exposed to this process were: ECE111 Introduction to C Programming (six domestic and fifteen international freshman students) and ECE347 (one international and five domestic rising seniors) an undergraduate cross-listed course with GECE547 graduate level Embedded Systems Design (all international).

For compliance to the standards, the goal was for all students to be 100% compliant (9-point avg.) by the end of the semester. Figure 4, shows that ECE111 average rubric scores improved through the semester and it shows where work is needed.

An observation could be inferred that the concepts of thoughtful and reasonable commenting, capturing of test plans, designing to requirements, and proper naming conventions were split along nationalities. The author needs to address this earlier and possibly add additional insight to assist international students.

ECE111 - Key Assignment Avg	Hw5	Hw7	Hw9	Hw10
Criteria				
Submission Process	3.7	3.7	6.7	7.2
Comments / Documentation	2.2	2.6	3.2	5.5
Naming Notations	2.1	2.1	4.3	5.1
Code Style	3.9	4.9	6.7	7.6
Compilation and Linking	6.0	8.3	8.3	9.0
Execution / Runtime	4.3	9.0	9.0	9.0
Meeting Requirements	3.1	3.1	6.3	7.2
Professional Relevancy	4.5	4.7	5.9	8.4

Figure 4: ECE111 Rubric Avg

The three sections in the ECE347/GECE547 course faired a little more differently in compliance evaluation. This was the first software related course with the author, for the graduate students in two sections for GECE547, and all of them were international. Figure 5, shows an unusual reluctance to Professional Relevancy, Meeting Requirements, and Naming Conventions. Additional time and a special lecture were provided to them before Hw5. Hw5 and the Project were multiple team assignments. These teams were made up of two smaller teams of two students each. These larger teams had fewer deliverables of code per student, so the sample size to evaluate compliance became smaller. The jump in scores from Hw5 and the Project was due to another lecture of team formation based on various business techniques experienced by the author. The main team ingredient for the Project assignment was a creation of a software quality and compliance role a student assumed. That student sole responsibility was the validation test plan and making sure the code was compliant. That lecture will be given again on all team building assignments for all software-oriented courses by the author.

GECE547 - Key Assignment Avg	Hw2	Hw3	Hw5	Project
Criteria				
Submission Process	2.4	4.7	8.4	9.0
Comments / Documentation	4.6	6.0	6.0	8.6
Naming Notations	2.1	2.1	4.3	9.0
Code Style	3.9	4.9	6.7	8.6
Compilation and Linking	6.0	8.3	8.3	9.0
Execution / Runtime	4.3	9.0	9.0	9.0
Meeting Requirements	3.1	3.1	6.3	8.6
Professional Relevancy	4.5	4.7	5.9	9.0

Figure 5: GECE547 Rubric Avg

The section in the ECE347 their overall scoring and compliance evaluations referenced in Figure 6, appear to be very inconsistent with the students in ECE111 and the graduates in GECE547. These students have had the author in numerous software-oriented courses. Many of the standards and processes were introduced in prior courses, except for Professional Relevancy, Code Style, and the *Programming Standards* document. They quickly understood the author's requirements and demonstrated the implementation in the next assignment. This indicates that consistent repetition helped enforce the concepts of following standards.

ECE347 - Key Assignment Avg	Hw2	Hw3	Hw5	Project
Criteria				
Submission Process	8.4	9.0	9.0	9.0
Comments / Documentation	7.0	8.0	8.0	9.0
Naming Notations	6.5	8.0	9.0	9.0
Code Style	3.9	4.9	6.7	8.6
Compilation and Linking	9.0	9.0	9.0	9.0
Execution / Runtime	9.0	9.0	9.0	9.0
Meeting Requirements	6.0	6.0	6.0	9.0
Professional Relevancy	3.0	3.0	9.0	9.0

Figure 6: ECE347 Rubric Avg

Once exposed to a standard and enforced compliance the student understands the fundamental concepts consistent in most industry standards. A colleague was concerned that the students would get use to one standard and it may be hard to start using a different standard in the industry. That was a good observation, understanding that there are many standards in the industry and by understanding the concepts, the student would experience common concepts with different applications. How to educate where variation between standards is another challenge. One possible approach is to vary the standard between class threads or semesters. The author will take this challenge bearing in mind that these standards are learned through application, not additional lectures.

Wernher Von Braun was attributed to the quote: "One test result is worth one thousand expert opinions." The author received his "One test result" as an email from one of the students in ECE111. "

"**...** *The interview was held at FMC Technologies in Erie, PA. There were two Software Engineers and a Head of Engineering Department.*

Q: What do you in your C Programming Class?
I answered:

- First of all, our teacher says it's not only a programming language course where students learn about the syntax and how to write a code, but also it's a designing course where students learn how to implement a good algorithm. I agree with him because from the beginning of the semester, we started from a simple task where we need to work on data and bound. As we moved forward, we developed our initial code into next level. By meaning, that every time when we learned a new topic, we did not leave our previous work there and started work on something different. Now at the end of semester, we have are working on a code which has included every topic that we covered.

Our teacher tries hard to teach us business world Coding Standard . When we receive a task to work on, we were told to follow the Coding standards. So we could experience the business world standard from the beginning and well prepared when we graduate from college. On the other hand, by following the Coding Standard, we can learn how to write a code, which is readable and reusable in the industry. The taught us how improve coding by doing peer review as well as following the standards when developed our previous code into next level.

The Software Engineers and Head of Engineers were impressed by that I have experienced with Coding Standards in the C Programming course. I think they really impressed by the fact that I have been told to learn and follow the Coding Standard in Freshman level in the college. One of the Software Engineers said it would be easier or simpler for them to work with and train me if I already had some experience of Coding Standard.

As a result of the interview, I accepted to do Software Engineer internship in the FMC Technologies. I am positive that the fact, which I have some working

experience in the class with Coding Standard, influenced them to make a final decision."

The Author takes the result that this pedagogical method can work and has merit to take our students to the next level, standing out in interviews and being more employable.

REFERENCES

[1] Dr. Lyle N. Long, "The Critical Need for Software Engineering Education", Crosstalk The Journal of defence Software Engineering, pp. 6-10, January 2008.

[2] DOD-STD-2167A, US Military Standard: Defence System Software Development (29 FEB 1988) [S/S BY MIL-STD-498].

[3] MISRA C (MISRA C:2012)Guidelines for the Use of the C Language in Critical Systems, ISBN 978-1-906400-10-1, ISBN 978-1-906400-11-8 (PDF), March 2013.

[4] Michael Barr, BARR Group, "Embedded C Programming Standard" barrgroup.com/coding-standards, 2013.

[5] Donald V. MacKellar Jr., "*Programming Standards*", availbale at http://ece.gannon.edu/public

[6] C. Boogerd, L Moonen, "Assessing the Value of Coding Standards: An Emperical Study", DOI: 10.1109/ICSM.2008.4658076 · Source: IEEE Xplore, Conference: 2008. ICSM 2008. IEEE International Conference on Software Maintenance

[7] International Organization for Standardization (ISO), reference: http://www.iso.org/iso/home/standards.htm.

[8] A Practical Handbook of Software Construction, Microsoft Press; 2nd edition (June 19, 2004), ISBN-13: 079-0145196705 ISBN-10: 0735619670

[9] Information on EvalTools® available at: http://www.makteam.com

[10] F. Mak, J. Kelly, "Systematic Means for Identifying and Justifying Key Assignments for Effective Rules-based Program Evaluation," 40th ASEE/IEEE Frontiers In Education Conference, Washington DC, October 2010.

[11] B. S. Bloom, Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain, New York: Longman. Jan 1984.

[12] R. Buse, W. Weimer, "Auotmatically Documenting Program Changes", ASE '10 Proceedings of the IEEE/ACM international conference on Automated software engineering, Pages 33-42