Design Inspections and Software Product Metrics in an Embedded Systems Design Course

J.W. Bruce
Mississippi State University

Abstract

Development tools, especially those for software, have matured to the point where a single iteration of the development cycle can be as short as a few minutes. No one desires to go back to the "good old days" when the development and physical prototyping cycles took hours or days. However, the slower development pace of yesteryear did prompt a certain amount of critical review of design changes and undoubtedly prevented many basic design defects. Current development tools combined with the increasing time-to-market demands lead engineers (and engineering students) to design at a frantic pace, often introducing many design defects. An easy way to improve the quality of design is to get the engineers to simply "slow down".

This paper describes a design process for an embedded systems design course [1] where formalized hardware and software design inspections are performed. The design inspections are held before prototyping begins and strives to curtail the far too common cycle of develop, test, change, and test again – a cycle I describe as "hacking". The design process described in this paper yields a high-quality product within a short design cycle, while mimicking the design inspections found in industry [2] [3].

The design inspections serve as a convenient time for software product measures to be collected. The quantitative measures document the nature, origin, and other vital characteristics of each design defect and are frequently used in industry [4] [5]. Furthermore, data obtained in design reviews can be used to improve the instruction quality, track the maturity of the student design skills, and prompt relevant classroom discussions. Examples of using the software product metrics in design process monitoring, analysis, and estimation are given.

Finally, the design practices described in this paper help students to develop team and communication skills that are often neglected by traditional engineering curricula. Course evaluations were obtained from students and external reviewers. Results indicate that the process is well received and achieves the course's educational objectives.

1 Introduction

In [[1]], the author presents a team-based progressive embedded systems design course that, in addition to providing the technical embedded systems knowledge, develops team and communication skills in situations emulative of industry. The course was a success by many accounts; however, student teams abandoned sound design practices in attempt to meet the demanding 16-week "time-to-market" constraint. Teams adopted a rapid development model where design defects are detected and corrected in unit and system testing. Designs were not reviewed other than *ad hoc* reviews by the designer. Consequently, team members produced

defect-riddled designs and the design schedules slipped due to an unproductive test-redesign-test development cycle. This paper introduces a lightweight design process based loosely on proven software engineering standards that strives to detect defects during design. This development process has been used with success in the subsequent offerings of the design course in [[1]]. Furthermore, the design process here allows the teams to collect product measures to monitor the students' design effort and development efficiency.

2 Development Process

In the university environment, the "time-to-market" is approximately 16 weeks. This short duration makes it difficult to motivate, train, and deploy a full-featured industrial software development process with students possessing a limited software engineering background. The foremost learning goal of the course in [[1]] is embedded systems design, including hardware, software, and interfacing. However, a secondary goal is to create a realistic team-based design environment while maximizing the design productivity of the student teams. To this end, two software engineering standards, IEEE Std. 1028 [[2]] and IEEE Std. 982.1 [[3]], were used to create a lightweight development process to detect design defects before testing and improve the probability of a successful design during an academic semester timeframe.

2.1 Inspection: Roles and Mechanics

IEEE Std. 1028 defines five different types of review for software development [[2]]. In the interest of time, only one review, the "Inspection", is used. The inspection in [[2]] has been adapted here for software and hardware. These inspections create a peer evaluation of a design before it is deployed. During the review, "impartial" reviewers (team members who are not author) identify defects, deviations from coding and design conventions, and design specifications. The design's behavior as written, not its intended behavior, is reviewed critically. Other software inspection styles can be found in [[4]]. The goal of the inspection is to identify as many design defects as possible. Design fixes are made by the author later.

In the proposed inspection process, there are four well-defined design review roles: coordinator, author, reader, and recorder. Ideally, each role is played by a different team member. The role of the coordinator is to facilitate communication, schedule meetings, and ensure the process is successful. The author is the person who authored the design and, ultimately, corrects the identified defects. The reader and recorder act as impartial reviewers, in addition to the coordinator.

After a design module has been authored, the inspection procedure has five steps:

- Planning (coordinator and author)
  After the designer finishes a design component, the author requests an inspection meeting be scheduled and sends the design to the coordinator. The coordinator determines a time and location for the meeting that is suitable for all team members, and forwards the design to each team member. For each design milestone described in [2], there might be several significant design components created by different authors.

- Preparation (entire team)

Each team member prepares for the meeting by reading and inspecting the candidate design before the meeting. The author is encouraged to do so as well. One study has shown that upwards of 90% of code defects are identified during the preparation step before the actual code design review meeting [5], so this preparation step is crucial to the success of the inspection process.

- Design Review Meeting (entire team)
  The coordinator's role is to keep the review meeting on task and to ensure attendance by all team members. The reader's role is to "read" the design's behavior aloud. The reader must be vigilant to paraphrase the design behavior as written, not its intent. Therefore, the reader should never be the author. The reading should be comprehensive; the review critiques everything about the design: software instruction, headers, comments, formatting, coding and design conventions, and interfaces. The recorder maintains written record of the findings of the inspection meeting. The designer is present only to clarify questions that cannot be ascertained from the written design itself. (Requests for the author to clarify are likely an indicator of poor design.) Under ideal conditions, the author will not be required to say anything during the design review meeting. The coordinator needs to ensure that the inspection is impersonal. The author and his/her ability are not critiqued, only the design is under review. The author should never be placed into a situation where he/she feels forced to defend himself/herself.

- Rework (author)
  At the end of the inspection meeting, the team determines whether the severity and number of defects warrants an additional design inspection. If so, the author uses the design review meeting documentation to correct identified defects. Then, the author submits the corrected design to the coordinator who schedules another design inspection. If defects are few and minor in nature, the team may opt for the author to correct the defects and proceed to the follow-up step.

- Follow-up (coordinator)
  After the author has corrected the defects identified in the design review meeting, the revised design is sent to the coordinator. Using the design review meeting documents, the coordinator determines if the author has corrected the defects satisfactorily. If corrections do not appear satisfactory to the coordinator, he/she can request additional rework by the author. When the coordinator is satisfied with the code, the coordinator approves the design to proceed to deployment and testing in the lab.

It is convenient to impose an additional constraint: inspections are done after the design is complete, but before testing. For hardware designs, the inspections are held after components have been selected and a schematic is generated. For software, inspections are held after the author has generated an error-free and warning-free compilation. Students are instructed that hardware cannot be prototyped and software cannot be simulated or tested on hardware before being certified in the inspection process described above. Tools like compilers are used only to check syntax. Design function is to be verified by the author mentally. Therefore, students are forced to understand their design thoroughly and verify every contingency. Too often, students (and mature designers, as well) design thinking if that some esoteric, unexpected behavior is created that it will be caught in testing. Much research has shown that correcting defects at later

stages of development is much more expensive than if corrected earlier [[5]]. Since simulation and testing are not allowed before peer review, the designer will spend the time to think through the design and its effects fully so as not to be embarrassed.

2.2 Product Measures

Software development metrics can increase productivity, identify development process shortcomings, increase software quality, and aid in development planning [[5]]. Student confidence can be increased by comparing their metrics to those published in the literature. Furthermore, the instructor gains from collecting metrics on the students' software process. The instructor can observe the improvement in individual and class abilities, as well as acquire indicators of the relative complexity of homework and design assignments.

Many software measures in IEEE Std. 982.1 are obtained very naturally during the development process described in the previous section. Throughout the design activities and the inspection process, students are asked to record

- Defects -- identified by author, design task, defect type, and severity
- Person-hours -- recorded by team member, task, and activity (design, review, deployment, or testing)
- Output -- lines of code (LoC) identified by author, design task, and software routine

Each team member maintains his or her own records, and is required to compute several additional measures. For example, coding efficiency is computed as LoC per workday, where a workday is defined as eight person-hours of effort, development costs per LoC are computed assuming a hourly rate of $75/hour, and a code quality measure is computed as number of identified defects per thousand LoC. Data collection is facilitated by forms for recording time spent in each activity, recording details of defects found, and summarizing inspection findings. A spreadsheet is provided to compute all measures by individual, team, and design task. Example data collection forms and spreadsheets can be obtained by contacting the author.

2.3 Use with Design Project in [1]

At several points during the semester, design teams are required to forward their measures data to the instructor. Individual and team data is disseminated to the class. This disclosure allows the instructor to (i) give frequent feedback to ensure quality data collection, (ii) identify teams with a poor team dynamic, (iii) promote a friendly competition between teams to operate with maximum efficiency, and (iv) motivate engaging classroom discussion on ethical, economic, and design method issues.

As might be expected, some students resisted the process described here as a "complete waste of time". Students argued that designers are "born, not created". Many examples from the literature to support quantitatively the effectiveness of development process were given in counter argument. Students are asked to follow the prescribed procedure for a few weeks. A promise to discuss, evaluate, and incorporate any suggested improvements usually sways stalwart resistors. (This is an excellent way to give students ownership and responsibility of their

own learning.) After the first design milestone, one team was elated to give a testimonial about how the process identified all of their design defects in the preparation step and design review meeting. The team declared the process valuable since their design worked upon assembly and downloading of their software. For the remainder of the semester, this team insisted on following the design procedures exactly. This team consistently finished assignments first and with high quality. Clearly, this team had internalized the development process and made it a part of their "professional persona". Throughout the semester, several other design teams reported similar experiences.

Remaining students skeptical of development process were converted not by the benefits of IEEE Std. 1028, but by the utility of the information in the measures of IEEE Std. 982.1. These students recognized that the development process described here is a convenient method to obtain the measures. The usefulness of the metrics is made apparent to them by a subject near and dear to everyone's heart - money. Each team is instructed to use their product measures to calculate coding efficiency, defect-production rate, and development time expended in order to determine a manufacturer's suggested retail price (MSRP) for the design. MSRP is calculated for several different potential sales volumes. Table 1 shows one team's MSRP calculations for 100, 10000 and 1 million units. Each team can see exactly how their productivity affects the bottom line. The exercise is an excellent vehicle for initiating discussion on the design process and the economic issues involved, including "bill of materials" costs; non-recurring engineering costs; manufacturing costs; facilities and administrative costs; distribution networks; etc. This exercise was more successful than could have ever been imagined. Students were fully engaged in discussion and immediately recognized that the age-old adage "time is money" is very much true. After this exercise, students appreciate that the collected measures can help them to monitor an ongoing design process and to make future design schedules that are more accurate.

Table 1: Manufacturer's Suggested Retail Price (MSRP) Calculation

Manuf costs % of BoM costs	100%	Manufacturing costs as percentage of BoM	
Manufacturer margin	35%	Manufacturer's profit (as percentage of TMC)	
Distributor margin	30%	Distributor's profit (as percentage of distributor's cost)	
Retailer margin	40%	Retailer's profit (as percentage of retailer's cost)	
NRE costs	$ 18,750.00	obtain from Team semester metrics	
Units sold (for amortization)	100	10000	1000000
Manufacturer			
Development cost/unit	$ 187.50	$ 1.88	$ 0.02
BoM cost (Given by instructor)	$ 48.30	$ 48.30	$ 48.30
Estimated manufacturing costs (labor, F&A on manuf., packaging, etc.)	$ 48.30	$ 48.30	$ 48.30
Total manufactured cost (TMC)	$ 284.10	$ 98.48	$ 96.62
Margin	$ 99.44	$ 34.47	$ 33.82
Manufacturer's sales price	$ 383.54	$ 132.94	$ 130.44
Distributor			
Margin	$ 115.06	$ 39.88	$ 39.13
Distrbutor's sales price (wholesale)	$ 498.60	$ 172.82	$ 169.57
Retailer			
Margin	$ 199.44	$ 69.13	$ 67.83
Retailer's sales price (MSRP)	$ 698.03	$ 241.95	$ 237.39

The analysis is easily modified to have students find coding efficiency required of their team to reach a target MSRP, defect production and detection rates required to reach a target MSRP, and time-to-market given target development efficiencies. The class also enjoyed an exercise inspired by "fantasy sports teams" where students would use each student's individual measures to form a development "dream team" to get shortest time-to-market, lowest development cost, or fewest defects. Finally, the development process and product measures described here give the instructor great freedom to discuss many of the "softer" aspects of engineering design.

3 Discussion

At the course conclusion, students are asked to take an optional online survey. The online survey was built, in large part, using the assessment infrastructure developed for EC2000 accreditation at the author's institution [[6]]. The survey questions the students about the frequency of selected actions, their perceived progress in abilities, and change in confidence levels. Table 2 shows the students' perceived frequency of performed several tasks. Results are ordered by average score with standard deviation given in parentheses. Frequencies are rated from zero (never) to three (very often/almost always). Students felt the design environment was "often" like what they thought industry would use. A significant portion of the students felt that they were "very often" called upon to work and communicate as a team and given assessment of each team member's contribution. Table 2 shows that the focus on the team-based development process is apparent to the students.

Table 2: Task Frequency Survey Results

In THIS COURSE, ... (N=15)	*Score (0 to 3)*
I had to make critical assessments of my work, other's work, and our relative contributions.	2.67 (0.617)
I had to learn to work and communicate with others to accomplish my tasks.	2.60 (0.632)
I used knowledge or skills learned in previous/other classes.	2.53 (0.640)
I had to supplement material learned in lecture with self-study.	2.20 (0.941)
the design environment is very similar to what I know/envision industry to be like.	2.07 (0.829)

Students rated their personal progress in each of the course objectives formulated in [[1]]. Responses are rated from zero (none) to three (a great deal). Table 3 shows the survey results ordered by average score. Students unanimously felt their understanding of embedded systems increased "a great deal". This learning objective constitutes the "information transfer" expected by students. However, it is interesting to note a large increase in appreciation for the value of design documentation. Experience has shown that students often dismiss these documents initially and express these feelings aloud. It seems that by the semester end, the students have bought into using these documents in the design process. As hoped, students responded that their abilities to learn on their own, work in teams, and troubleshoot were improved by the course. Time constraints prohibited many of the planned team-building exercises, however, students note "some" improvement in their ability to motivate their teammates.

Table 3: Learning Objective Progress Survey Results

Progress made, BECAUSE OF THIS COURSE, in your ... (N=15)	*Score (0 to 3)*
understanding of embedded systems and components	3.00 (0.000)
your appreciation of the usefulness of design documentation	2.80 (0.414)
ability to create a system to meet a detailed design specification	2.67 (0.488)
Skill to work in an engineering design team	2.60 (0.632)
ability to evaluate and justify competing designs against a specification	2.40 (0.737)
Skill in assessing the performance of yourself and others	2.00 (1.000)
ability to motivate others to achieve their maximum potential	1.67 (1.175)

4 Conclusions

In order to meet the need for embedded systems engineers, the author has created an embedded systems design course that emulates industrial situations as much as possible. The student teams build a progressively more complex design using formal and documented design reviews and

collect product measures to monitor their design performance. Design teams detect defects in the design phase rather than the more costly, and less efficient, testing phase. Students indicate an increased engagement and perceive an increased relevance of their education. Students also report increased confidence in team, communication, and lifelong learning skills.

References

[1] Bruce, J.W., Harden, J.C., and Reese, R.B. "Cooperative and Progressive Design Experience for Embedded Systems", *IEEE Transactions on Education*, vol. 47, no. 1, pp. xxx-xxx, Feb. 2004. (In press)

[2] IEEE Std. 1028-1997, *IEEE Standard on Software Reviews*. Section 6.

[3] IEEE Std. 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*.

[4] Gilb, T. and Graham, D., *Software Inspections*. Addison-Wesley, 1993.

[5] Pressman, R.S., *Software Engineering: A Practitioner's Approach 5/e*, Mc-Graw Hill, 2001.

[6] Harden, J.C. and Lane, M.G., "Web-based tools for assessment automation," *Proc. ASEE Annual Conf. and Expo.*, Session 1532, 2002.

Biographical Information

J.W. Bruce received the B.S. degree from the University of Alabama in Huntsville in 1991, the M.S.E.E. degree from the Georgia Institute of Technology in 1993, and the Ph.D. degree from the University of Nevada Las Vegas in 2000, all in Electrical Engineering. Dr. Bruce has served as a member of the technical staff at the Mevatec Corporation and the Intergraph Corporation. Since 2000, Dr. Bruce has been with the Department of Electrical and Computer Engineering at Mississippi State University, where he is an Assistant Professor. Dr. Bruce teaches courses on embedded systems, VLSI, and systems-on-a-chip design and was named the Bagley College of Engineering Outstanding Engineering Educator in 2003. Dr. Bruce researches data converter architectures and embedded systems design. He is the author or coauthor on more than twenty journal articles or technical publications. Dr. Bruce served as Associate Editor of *IEEE Potentials* from 1997-2001. He is a member of Eta Kappa Nu, Tau Beta Pi, and ASEE.