Bridging the Gap Between Software and Non-Software Engineers

An Online Continuing Education Course for Engineers

Course Number: IC-2005

Credit: 2 Hours / 2 PDH / 2 CPD

© 2023 Decatur Professional Development, LLC. All rights reserved.

Bridging the Gap between Software and Non-Software Engineers

Cheng-Ning Jong, P.E. and Tracy P. Jong, Esq.

Table of Contents

©2011 Tracy P. Jong and Cheng-Ning Jong. Portions ©2011 Decatur Professional Development, LLC All rights reserved

Introduction

The days of pure mechanical devices are quickly fading. When you look around, every device seems to be automated in some fashion. Even the old fashioned hammer and screwdriver have popular electric versions! You probably own one yourself.

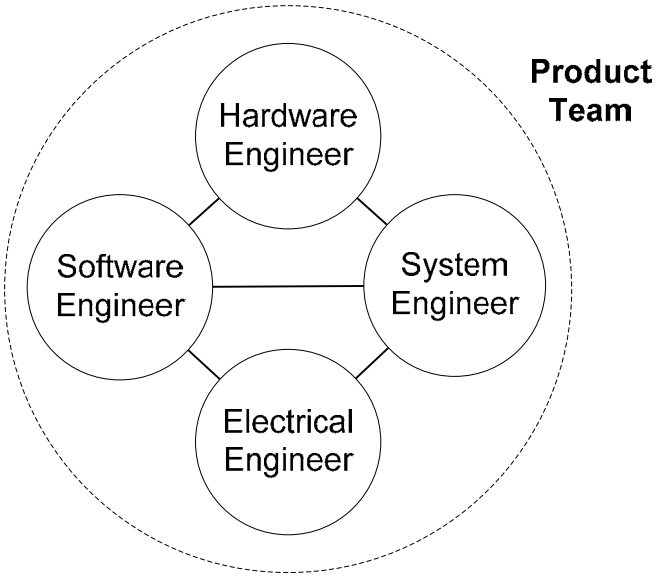
The simplest of devices often interfaces with systems running on software or contains microcontrollers which have software running on them. As computer use has become widespread, the cost of incorporating software into control systems, analysis, project tracking and management systems continues to decrease. Software has become more complex, however and user interfaces have become increasingly user-friendly as computing power increases.

Every product that we use began its life in a manufacturing environment. Let's look at the dynamics involved in manufacturing a simple electronic device. No man is an island. Today, a typical product project team comprises

- mechanical/structural/hardware engineers and designers who design the physical components that make up the product;
- electrical engineers who design the controller/s and write firmware that control the individual physical components;
- software engineers who design and write software to synchronize, schedule and control the components as a system; and
- System engineers who work closely with all parties to design and solve problems arising from interactions between components.

The addition of software component in a product requires a team approach to product development and production. There must be effective communication between software engineers and hardware, electrical and other non-software engineers. The hardware, electrical and other non-software engineers generally provide the "*requirements*" to the software engineers. Only with effective communication between these parties will software engineers be able to develop and implement reliable software.

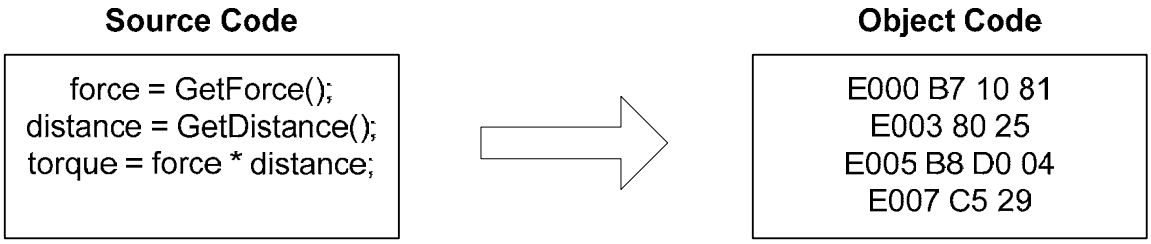
©2011 Tracy P. Jong and Cheng-Ning Jong. Portions ©2011 Decatur Professional Development, LLC All rights reserved



Good software engineers focus on the opportunity to minimize "rework" in future product iterations and upgrades. Well-written software code often contains generic features and maximizes opportunities to reuse and re-factor without complete redevelopment of the code content and algorithms. However, some applications do not lend themselves to generic coding and requirement owners should not be easily swayed when confronted by the pressure to conform to such generic coding. Understanding the basics of software will enable the software developers and requirement owners to effectively develop coding applications.

©2011 Tracy P. Jong and Cheng-Ning Jong. Portions ©2011 Decatur Professional Development, LLC All rights reserved

How does a line of code turn into machine action?

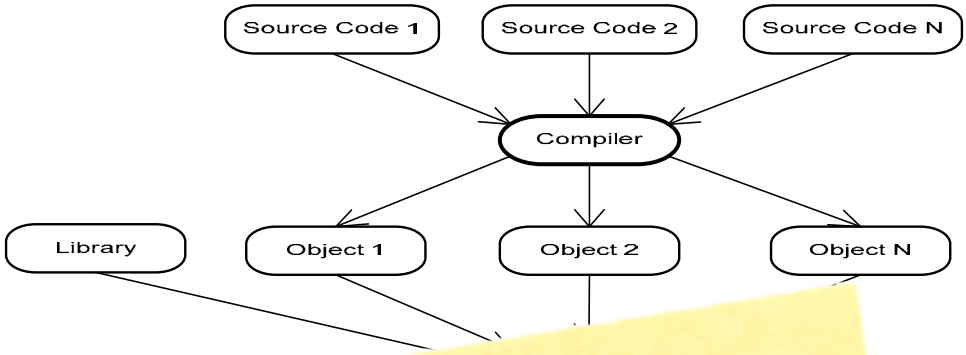


A line of code can take on various forms depending on the compiler utilized. A *compiler* is a software program that takes text written in a computer language called *source code* and translates it into another form of computer language called *object code* or simply *object*. A compiler requires the source code to be written in certain syntax so that it can be properly processed.

Even though there are many commercial *Integrated Development Environment (IDE)* tools available to facilitate software development, a simple *text editor* is all that is required to write code. A basic IDE incorporates a source code (text) editor, compiler, linker and debugger. The function of a *source code editor* is very much the same as a normal text editor such as in a typical word processing program. However, a programmer need not be concerned with tab, indentation, margin setting and the like because the compiler's parser is typically designed to handle such details. A *linker's* function is to take one or more objects generated by the compiler and assemble them into an executable program. A *debugger's* function is to facilitate verification and debugging of the source code. Upon debugging, the source code may be relatively free from syntax errors but there is no guarantee that the resulting source code performs according to product requirements.

The following diagram depicts the process of generating an executable file from source code.

©2011 Tracy P. Jong and Cheng-Ning Jong. Portions ©2011 Decatur Professional Development, LLC All rights reserved



To view the remainder of the course material and to take the quiz for PDH credit, you must purchase the course.

Close this window and click "Add to cart" on the product page.

How (' The compiler lexica ishes relatio builds object code. (which will be explain or machine languag the "Compil ed by the "Linker'

Source c am of software superior as a tool t high level computer mbly language a ires no interpretati comes in a series of tly, however, it memory ad chine code instruc

A high-level higher level of abstraction than a lower-level language. For instance, a higher level language would have human legible syntax and be more compact in its functionality. It frees the software engineer from having to worry about interactions between machine code and computer hardware, such as input/output (I/O) registers, memory management, rudimentary calculations, etc. In comparison, the use of a low-level language such

©2011 Tracy P. Jong and Cheng-Ning Jong. Portions ©2011 Decatur Professional Development, LLC All rights reserved