# A Framework for Developing Graphically Programmable Low-cost Robotics Kit for Classroom Education

Mehedi Hasan Masum
Dept. of CSE
University of Dhaka
mehamasum@gmail.com

Tanvir Shahriar Rifat
Dept. of CSE
University of Dhaka
rifat.csedu20@gmail.com

Saifuddin Md. Tareeq
Professor,
Dept. of CSE
University of Dhaka
smtareeq@cse.du.ac.bd

Hasnain Heickal
Assistant Professor,
Dept. of CSE
University of Dhaka
hasnain@cse.du.ac.bd

## ABSTRACT

The purpose of this paper is to present a project that provides a framework to build educational robotics kit with low cost components and interface the kit with a visual programming language. A robotics kit is inaccessible to many third world schools due to its high cost. This paper provides a “Do It Yourself” (DIY) approach to produce graphically programmable robots with low cost components. The framework consists of the low-cost hardware components, back-end software and the visual programming editor to interface the hardware. The editor consists of interlocking graphical blocks to represent programming concepts such as Variables, Logical expressions, Conditions, Loops, Lists, Event Listeners, Parallel Programming and many more. Using this editor a student can graphically reprogram the robotics kit firmware and manipulate it. It allows young adults to apply programming concepts without having to worry about the syntax and makes programming easy and fun.

## CCS Concepts

**• Applied computing→Computer-assisted instruction•Applied computing→Interactive learning environments**

## Keywords

Visual programming language; Low-cost robotics kit; Educational robotics; Classroom education

## 1. INTRODUCTION

In today's fast paced and ever changing world, the domain of pedagogy needs to strike a balance between traditional teaching methods and the dynamic technological tools. Technology is capable of positively influencing our learning and thinking abilities. The natural human tendency of exploring and inspecting can be intensified through the introduction of educational robotics into a curriculum at primary [1], secondary, undergraduate [2] and even postgraduate levels [3]. Programming a robotics kit in classroom environment is a highly stimulating activity for a young learner, thus he might be more interested in classroom education. The process of providing immediate feedback to a learner is very important in education. When a learner programs the robot to move in a certain way and then immediately sees the results he wanted, he knows that he has manipulated the code correctly. This is a very powerful source of instant positive reinforcement. We believe that introducing this sort of programmable robotics kit in the classroom environment will attract more students to the classroom and may significantly help preventing school failures and drop outs.

According to [4], robotics kits have been divided into five categories: Building Body Kits, Electronic Components, Software Kits, Programmable Robots, and Complete Starter Kits. Our focus is on Programmable Robots (specifically the graphically programmable kind) where the kit offers no flexibility in terms of hardware or electronics expansion but allows the user to reprogram the firmware with the help of programming. In this regard, Visual Programming Languages help users to apply programming concepts without having to worry about the syntax or the intimidation of a blinking cursor on a command-line.

A robotics kit is inaccessible to many third world schools due to its high cost. This paper provides a “Do It Yourself” (DIY) framework to produce graphically programmable robots with low cost components and design them in such a way that they can be easily built by a team of undergrad students.

In the next section, the main framework along with its components are discussed. Section 3 elaborates on the details of an implementation of this framework. Section 4 contains results and analysis of the whole project. Finally, we conclude with a small discussion and acknowledgements.

## 2. THE FRAMEWORK

### 2.1 Overview

The principal behind this framework is simple- make use of low cost hardware components to build up a mobile robotics kit and exercise interoperability with libraries and open-source frameworks to make it graphically programmable by the means of a visual programming language editor. The robot firmware is reprogrammable in real time via the graphical blocks of the visual programming editor.

There have been some other works [5, 6] on developing visual programmable robotics kit. Most of these products are costly and not easily reproducible. They are not designed analyzing the

*ICETC 2018*, October 26-28, 2018, Tokyo, Japan.
© 2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6517-8/18/10…$15.00
https://doi.org/10.1145/3290511.3290535

context of a developing country where smartphones or heavy duty workstations are not widely available, Internet is slow & unreliable, and even electricity is still an issue. So we designed a DIY framework keeping those issues in mind and restricted the building cost to a certain range so that the final product is easily affordable for schools. We introduced advance features like Event Handling and Parallel Programming which have not been used in other similar projects and also left enough room for improvisation to incorporate exciting new features, for example, applications of robotic vision.

We will focus on the design goals before diving into architectural details. Hardware specific design goals are: reproducible kit with cheap components, easy assembly and setup, free movement of robot body with sense of environment, low power consumption and real time programmable firmware. Software specific design goals are: no installation requirement, no specific device or OS requirements and no Internet requirement. The framework is built having these goals in mind.

The framework comprises of two parts: the hardware (i.e. robotics kit) and the software (i.e. visual programming language & driver programs) as shown in Figure 1.

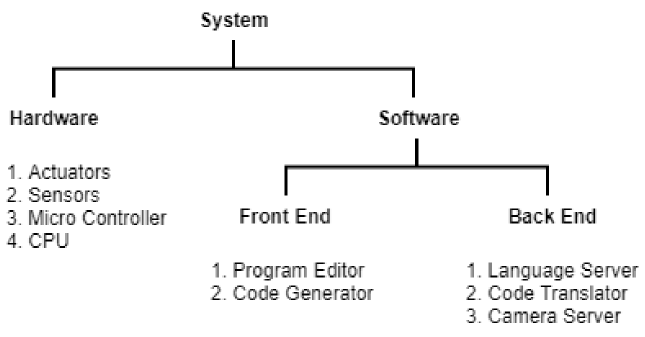


**Figure 1. The overview of the framework.**

The hardware consists of the actuators, sensors, a micro controller and a lightweight CPU. The software comprises of two parts namely: the front-end (visual programming editor) and the back-end that comprises of HTTP server and a code translator. A camera server is optional.

## 2.2 Hardware

To make the kit easily movable, a lightweight robotics chassis is essential for building the main robot body. First of all, it has to have supporting wheels that would be controlled by gear motors. Furthermore, the motor would have to be controlled by a motor driver which itself must be connected to a mother device so that it can be easily reprogrammed using the visual programming editor.

This mother device needs to be connected to some sensors that will help the robot to perceive the environment. For example: the robot understands how far it is from an obstacle with an Ultrasonic Sonar Sensor, it understands if it is following a line or not by Reflectance Array Sensor, a Camera will act as a visionary sensor etc. The motors and the mother device must contain a rechargeable power source which will be embedded on robot body. Optionally it can have an OLED Display or a Speaker attached to its body for output purposes. This configuration is shown in Figure 2.

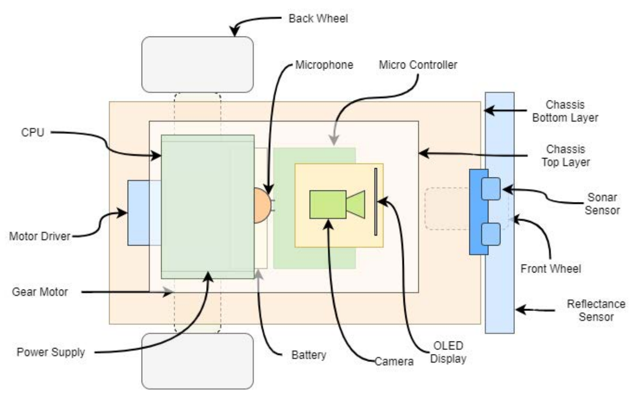


**Figure 2. Hardware buildup of the robotics kit.**

All these parts can be simply assembled together via simple jumper wires or a Printed Circuit Board (PCB). These hardware parts need to be interfaced via a web based visual programming editor. So a CPU is required to serve the visual editor over HTTP and connect back and forth between the client browser and the robotics kit. Thus a CPU is embedded on the robotic kit. This CPU will play the role of the mother device in this system. So it will be responsible to control the microcontroller and subsequent hardware pieces.

## 2.3 Software

The software architecture is rooted on the Operating System of the CPU. A HTTP Server is required to serve the web based visual programming editor. It serves the editor from the local file system. The served files are rendered on the user's browser as the Visual Programming Language (VPL) Editor. The VPL Editor consist of the graphical code blocks. Manipulation of them creates the user written program to control the robot. This program is be sent back to the server asynchronously by the editor.

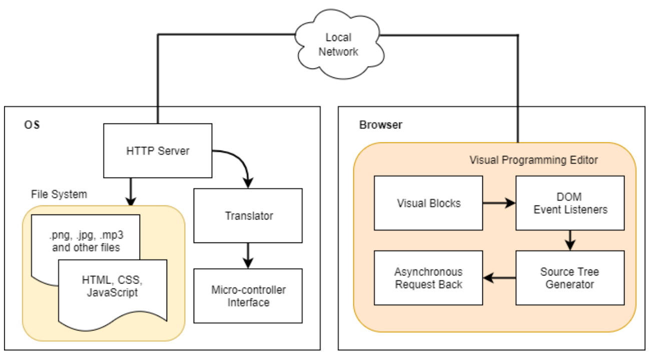


**Figure 3. Software architecture of the robotics kit.**

The program is then to be translated into lower level code by a Translator program in the CPU. The Micro-controller Interface is used to inject the translated code into the micro-controller so that it can control the robot's motors and other sensors. This process is visualized in Figure 3.

## 2.4 Interaction Model

The interaction model of this framework describes how the user interacts with the robotics kit. The CPU of robot creates a network Access Point during boot up. The user connects his device to the network just created by the robot so they are in the same local network. The address and security credentials of this network are previously made known to the user. Now the user opens up a browser in his device (PC, tablets or mobile phone) and connects to the server. The address of this server is also previously known to the user. The server on robot's CPU sends the visual programming language editor in response. This editor is designed in such a way that upon receiving the programming editor, the user can write programs by dragging and dropping graphical blocks onto his workspace in browser window. Then user hits the "Run button" and the Editor sends the program back to the server.

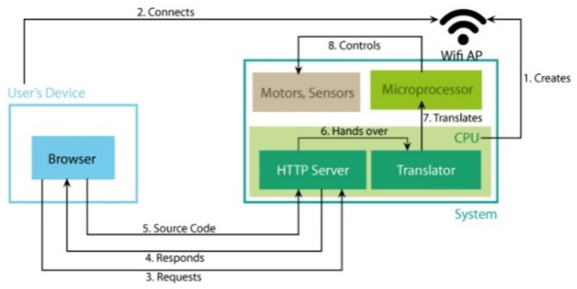


**Figure 4. Interaction model of the framework.**

The source code received from the user is translated into a lower level microcontroller code by the Translator in CPU. The microcontroller then runs the robot in low level i.e. controls motors, polls readings from sensors. Thus the robot runs according to the program written by the user. This flow is described in Figure 4.

## 3. IMPLEMENTATION

### 3.1 Overview

To implement the framework we provided above, we built a graphically programmable robotics kit named *Rupai*. It can move around, perceive the environment with sensors, display graphics on its screen and user can program it via a graphical editor to make it perform various actions.

### 3.2 Hardware Kit

We used an Arduino Board as the microcontroller of the robotics kit. Different sensors i.e. Ultrasonic Sensor, Reflectance Array Sensor and other low level kits i.e. Motors, Motor Shields etc. are connected to the Arduino Board and can be polled via the Arduino. The actions of the Arduino is controlled by a Raspberry Pi CPU, which acts as the master (mother device) in this system. A Linux based operating system, Raspbian is installed on Raspberry Pi. A WiFi Hotspot (Access Point) is set up in the Raspberry Pi. Devices will connect to this Hotspot network so that they are in the same local network. These hardware parts are fairly cheap and available around the world.

Here is the list of all hardware kits that were used: an Arduino Uno Board, two standard 12V Gear Motors, a Motor Driver (L293D), two Wheels (7 cm diameter), a LIPO Battery (12V), a battery charger, an Ultrasonic Sensor (HCSR04), a Reflectance Sensor Array (QTR-8A), a Raspberry Pi 3 Model B as the CPU, a Micro SD Card (32 GB) as storage device, a Power bank (10000 mAh) for powering the CPU, a Monochrome OLED Graphic Display (SSD1306), a Mini Camera/Video Module (OV5647), a Microphone, four Mini Breadboard, 2 sets of Jumper Wires and a lightweight Robotics Chassis. The OLED display is used to show information such as network address and security credentials to the user. Figure 5 shows the assembled hardware parts.

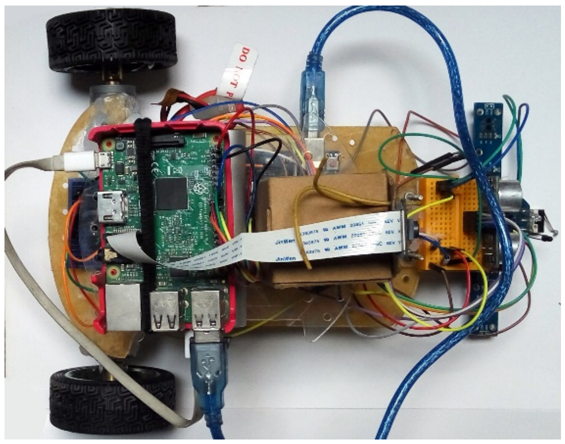


**Figure 5. After all parts are assembled together.**

### 3.3 Software

As the Visual Programming Editor will run on browsers, a Python HTTP Server is set up to serve the HTML & JavaScript codes on the Raspberry Pi CPU. Devices from the same network will send GET requests to the root endpoint and the VPL editor will be served in response. This Python HTTP Server also declares another end point to where the visual programming editor will send the program written by the user via a POST request. As a camera is directly connected with and controlled by the Raspberry Pi, a PHP server continuously polls data from the camera and sends back live camera frames to the editor when requested.

#### *3.3.1 Visual Programming Editor*

The programming paradigm of our visual programming language is inspired by Scratch from MIT Media Lab Lifelong Kindergarten Group [7]. Each robot functionality can be addressed by a graphical block. The blocks are interconnected and will be executed as a program by the robot according to their order (sequential operation). Our visual programming editor is web based and built on top of an open source library from Google named Blockly [8].

##### *3.3.1.1 Block Categories*

Categories of graphical blocks may vary according to different implementation requirements. As we can see in Figure 6, these are the top 11 categories for *Rupai*:

1. Action: contains blocks for all actions the robot can perform, for example driving the robot forwards, turning to specific directions etc.
2. Control: holds blocks that control the flow of execution of the program i.e. different Loops, Threads etc.
3. Logic: contains the logic blocks together: i.e. If-Do, comparisons, Boolean operations etc.
4. Math: contains trivial math operations.
5. Text: works with Strings and lets the user manipulate them.
6. Lists: lets the users create Arrays and manipulate them in code.
7. Sensors: used for reading sensor data and using them in code logic.
8. Events: allows users to create Event Handlers i.e. perform some functions on occurrence of some events.

9. Display: lets the users draw texts and geometric shapes in the OLED Display of the robot.
10. Variable: lets the users create Variables and work with them.
11. Functions: allows user to write Functions here and use them in the program.

#### *3.3.1.2 User Interface*

The User Interface of the visual programming editor has three main tabs to switch between three main views: Blocks tab holds all the categories of blocks of the visual programming language and the coding workspace, Code tab shows the live source code from the workspace, Feed tab polls live camera feed from the PHP camera server right in the browser. It has four main menus: "Save as File", "Open File", "Delete All Blocks" and "Run"- all of them are self-explanatory. "Run" sends the newly written source code to the Raspberry Pi server to run the robot.

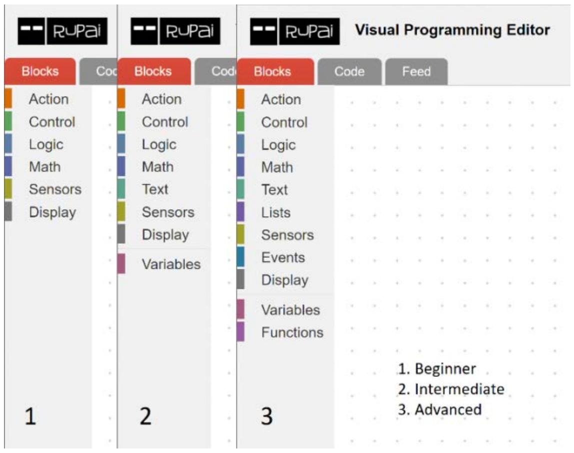
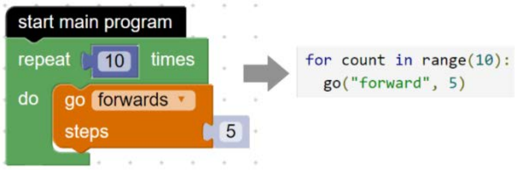


**Figure 6. Different Block categories for different user group.**

A user can choose between Beginner, Intermediate and Advanced programming level as shown in Figure 6. Beginners will have a much easier set of tools while the range gets wider towards intermediate and most wide for advanced users. "Language Settings" dropdown will change the language of the VPL Editor dynamically.

#### *3.3.1.3 Source Tree and Code Generation*



```
for count in range(10):
  go("forward", 5)
```

**Figure 7. Conversion of a Block to executable Python.**

As the backend is written in Python, the system requires the user program to be translated into an executable Python script as shown in Figure 7. This action is performed on the client side by Blockly. Though challenges still remain to strip out the event handlers from the main code to run them on a separate thread (dedicated to event loop) on the backend of the system. In order to do that, the Document Object Model (DOM) tree created by the Blockly is traversed with a Depth First Search and the branches that are related to the event handlers are taken out. Now the front end can send main programs and event callbacks separately to the backend for lower level translation.

### *3.3.2 Backend Scripts*

A HTTP server is created with Python to serve the VPL Editor as a web app, receive back the program written by the user and prepare it to inject in microcontroller. Python is used for most of backend scripting and automation purpose.

#### *3.3.2.1 Lower Level Translator*

After getting the user programs from VPL Editor, translating from the Python script to Arduino equivalent calls begins. For this task an open source library is used named *Nanpy* [9]. It uses Arduino as a slave, controlled by a master device, in this case-Raspberry Pi, where the Python scripts are run. *Nanpy* treats Arduino as an object as in Object Oriented Programming paradigm. Then this object is used to manipulate Arduino, e.g. to give appropriate digital value to its pin. In this project, Arduino controls the motors, fetches value from different sensors and so on. Figure 8 shows how the go("forward", 5) call of Figure 7 resolves to a moveForwards(steps) call.

```
begin function moveForwards(steps) returns null
    set speed of the left motor
    set speed of the right motor
    set rotation of the left motor to predefined value
    set rotation of the right motor to predefined value
    sleep for 0.2*steps seconds
    return
end function
```

**Figure 8. Low level translation of the program of Figure 7.**

#### *3.3.2.2 Event Handling*

Events are handled in a dedicated thread and are often used for polling sensor values from Arduino via Python because the primary events were mainly sensor based (i.e. "when obstacle detected" event seen of Figure 9). As the frontend strips out the event listeners callbacks from the main program, it becomes fairly straight forward to start a Python Thread and call the callback functions whenever an event occurs.

#### *3.3.2.3 Parallel Programming*

We allow our experienced students to take advantage of parallel programming via multithreading. Other graphically programmable robots available in market do not support multithreading, users can only code sequentially. Implementation of this feature is very similar to Event Handling and done via Python. Figure 9 shows an example program that takes advantage of the "execute in parallel" block to run two tasks at the same time.

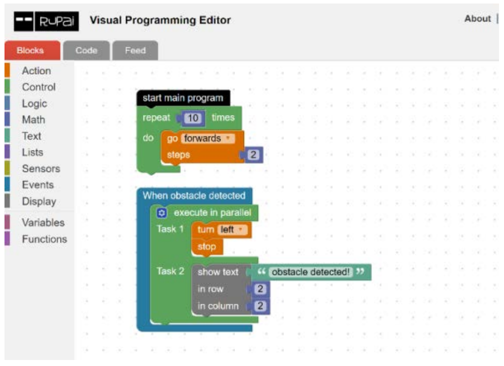


**Figure 9. Example parallel tasks written by user.**

### 3.3.3 *Advanced Feature Set*

By design the framework is built on top of a small CPU, therefore many advanced features can be implemented i.e. image/audio processing. In *Rupai* we implemented an object recognizer with the camera used in the robotics kit and also added a voice command listener as a part of our Event Listener interface. The implementation of those features are out of the scope of this paper

## 3.4 Potentials of Rupai in Classroom Education

The primary purpose of such a system is to teach programming through positive reinforcement in the classroom. However, there are many use cases in which *Rupai* can serve useful purposes. Some potential use cases of *Rupai* include:

1. Teaching Programming: The system allows user to write program without having to worry about syntax, so they can focus on the basics. Expert users can dive into some of the advanced concepts as Parallel Programming and Event Handlers.
2. Teaching Robotics and Hardware: The programming paradigm can be made of trivial robot hardware manipulation blocks. For example: “go forward” function can be composed of “write value to hardware pin” etc. User can easily learn to build a line follower or maze solver robot using the existing framework.
3. Teaching Science and Math: It can also be used to teach math, logic and other scientific ideas in STEM classrooms to improve problem solving skills of the students. Some simple examples can be measuring distance, angels, working with geometrical shapes etc.

## 4. RESULTS

*Rupai* is an example of the successful implementation of our proposed framework. Every piece of hardware used is easily accessible and cheap in price. The software part is mostly built on top of Open Source libraries and thus easily producible. Table 1 shows a comparison on different contexts among the DYI Rupai project and other commercial projects developed worldwide.

As the comparison suggests, Rupai is the only programmable robotics kit that let user take advantage of Event Handling and Parallel Programming. The framework is designed in such a way there exists a clear Separation of Concerns (SoC) [10] in both Hardware and Software part. For example, the CPU of the robot is only concerned with the communication between server and client, while the microcontroller is the one who manage lower level components such as motors and sensors. Another important aspect of the design of this framework is that the system has been designed in a portable way without any OS dependencies or any other system dependencies. Also this framework has included a fairly powerful CPU so that many exciting features which require CPU heavy operations (for example a Machine Learning Application) can also be featured in the robotics kit.

**Table 1. Comparison of graphically programmable robots**

| **Context** | **LegoMindstorms EV3** | **Dash, Wonder Workshop** | **Rupai** |
|---|---|---|---|
| Based on | LabVIEW | Blockly | Blockly |
| Prog. Paradigm | Flow based | Procedural | Procedural |
| System requirement | Desktop software | Smartphone app | Any web browser |
| Internet | Required | Required | Not required |
| User’s source code displayable | No | No | Yes |
| Event Handlers | No | No | Yes |
| Parallel Prog. | No | No | Yes |
| Cost | 400$+ | 200$+ | 100$+ |

*Rupai* is now currently in the prototype phase and being used as an educational robotics kit in some schools in Bangladesh. The feedback and analytical data of this pilot run will be helpful to take this study further ahead.

## 5. ACKNOWLEDGMENTS

We thank Department of Computer Science and Engineering, Robotics Lab of CSE, DU and the Ministry of Education, Govt. of the People's Republic of Bangladesh for their support.

## 6. REFERENCES

[1] C. W. Chang and J. H. Lee and P. Y. Chao and C. Y. Wang and G. D. Chen. Exploring the Possibility of Using Humanoid Robots as Instructional Tools for Teaching a Second Language in Primary School. Educational Technology & Society. 13:13-24. 2010.

[2] R. J. Wood. Robotic manipulation using an open-architecture industrial arm: a pedagogical overview [Education]. *IEEE Robotics & Automation Magazine*. 15(3):17-18. 2008.

[3] B. Moulton and D. Johnson. Robotics education: a review of graduate profiles and research pathways. *World Transactions on Engineering and Technology Education*. 8(1):26-31. 2010.

[4] A. R. Hilal, K. M. Wagdy, A. M. Khamis: A Survey on Commercial Starter Kits for Building Real Robots. *Proceedings of the International Conference on Electrical Engineering*. 2007.

[5] The LEGO Group. 31313 Mindstorms EV3 - Products - Mindstorms LEGO.com. 2013. Retrieved May 2, 2018 from https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313.

[6] Wonder Workshop Inc. Wonder Workshop | CleverBots for kids of all ages. 2013. Retrieved May 2, 2018 from https://www.makewonder.com.

[7] Wikipedia contributors. Scratch (programming language) - Wikipedia, The Free Encyclopedia. 2017. Retrieved December 26, 2017 from https://en.wikipedia.org/w/index.php?title=Scratch_(programming_language)&oldid=816846576

[8] Google Developers. google/blockly: The web-based visual programming editor.2011. Retrieved May 2, 2018 from https://github.com/google/blockly.

[9] A. Stagi. nanpy/nanpy : Use your Arduino board with Python. 2012. Retrieved May 3, 2018 from https://github.com/nanpy/nanpy

[10] Wikipedia contributors. Separation of concerns - Wikipedia, The Free Encyclopedia. 2018. Retrieved May 26, 2018 from https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=836253068