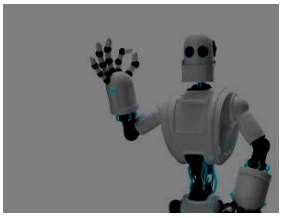
Bringing It Together
Robbie Robot Shop



CSE 1325 – Spring 2017 – Homework #7-#12
Sprint material is due every Sunday at 11:59 pm

One-week projects are fairly rare in the “real world”, though not extinct. “Short” projects take many weeks, while longer projects can run into months or years. We can't squeeze a multi-year project into our class, but multi-week – that we can do! In this homework, you'll apply the simplified Scrum process you practiced with our Library Management System project to manage a 6-week development of a simple robot shop management system.

As before, we’ll get started by coding the Model for a basic command line interface, then add a graphical user interface Controller and View. Feel free to adapt your earlier homework submissions, or the provided suggested solutions, in developing your project. Also note that, for the first time, you are *permitted but not required* to work on a self-selected team of up to 4 students for the duration of this project.

Introduction

Robbie Robot Shop (RRS) is a fledgling start up in the exciting field of robotics. They are seeking bright young programmers to build a custom solution for defining new robot products using their growing patented line of modular components, tracking customers and the robots they buy, rewarding their sales people well enough to keep them motivated, and otherwise taking care of business.

Your job is to win this project from RRS by producing a proposal package, including a prototype that wows and other creative and persuasive artifacts that prove you know your stuff. You have exactly 6 weeks to deliver RRS Manager (Prototype) v1.0 with your proposal package, at first featuring a glorious text menu or command line interface (CLI), and then finally a thoroughly compelling Graphical User Interface (GUI).

Overview

Robbie Robot Shop assembles their robots from 5 different components – a torso, head, arm(s), locomotor, and battery(ies). A Product Manager (PM) defines new robot models from these components, assigns a product name and price, writes up a brief sales description, and approves the result. The robot then appears at their Point Of Sale (POS) systems so that their Sales Associates (SA) can arrange for their Beloved Customers (BC) to order them, as well as on their web store so that BCs can order them direct. The Pointy-haired Boss (PB) will need to track profit margins, sales, and such from the robot product lines to ensure the business is profitable.

Requirements

The Product Owner has identified the needs of each of the actors that will interact with the system, and has assembled these into a prioritized Product Backlog in the Scrum spreadsheet, which is included in this set of Requirements by reference.

Product Manager (PM)

The PM needs to be able to create new instances of robot components in the system. For each component, they need to specify a name, part number, type (torso, head, arm, locomotor, or battery), weight, cost, and a brief description. In addition, some types of components need additional data: each torso may have from 1 to 3 battery compartments, each locomotor should specify a maximum speed (in MPH) and power consumed when operating (in watts), each arm should specify power consumed when operating (in watts), and each battery should specify the energy it contains (in kilowatt hours).

The PM needs to be able to define new robot models by selecting one or more of each type of robot component. Only one component of each type may be added, except that up to 2 arms and as many batteries as will fit into the selected torso may be attached. For each robot model, the PM also needs to define a model name, model number, and price. They will want to know the total cost of all components when setting the price to ensure adequate profit for each model.

Beloved Customer (BC)

A BC needs to be able to browse the catalog of robots, with pictures (eventually – not expected on the command line version), prices, shipping costs, and description at the POS terminal and (eventually) on the web store. Each customer will also need to view their orders and their outstanding bill.

Sales Associate (SA)

A Sales Associate (SA) needs to order robot models for Beloved Customers and present a bill of sale listing the order number, date of sale, customer name, robot(s) ordered, and price (subtotal, shipping, tax, and total). Each Sales Associate would like a sales report for the sales they personally completed with which to lobby for a raise.

Pointed-haired Boss (PB)

The Boss needs to know overall shop metrics, such as the profit margin on each robot and how many were sold, a complete list of orders during a specified period, and sales for each Sales Associate with which to approve or deny requests for raises.

Process

Each student developer will continue to follow the simplified Scrum process introduced for and practiced with Homework #4-#6, but with a complete Product (or Feature) Backlog that covers the full anticipated scope of the project – all 6 weeks.

As before, the developer is free to negotiate changes to the Product Backlog features and priority with the Product Owner (the Professor or TA). An approved change order is still required, an email from the Product Owner clearly stating the the change in the Product Backlog is acceptable.

You must provide your own *updated* UML models and design your own menus – that's part of this assignment. A basic, *incomplete* UML class diagram is provided to assist your understanding, but you must provide a complete diagram yourself. **Update your UML design before you start coding** – that's just good OO! **Neither the professor nor TA will discuss your project with you *at all* unless you bring your own UML models and up-to-date Scrum spreadsheet with you.**

Every class documented in the standard C++ library remains available for your use on this project, however, **you must use FLTK 1.3.x to implement your GUI**. You may also consider third party libraries that have been approved in advance by the Product Owner in writing, as long as you **comply with all license agreements.**

Students who team (see below) are expected to remain on their team for the duration of the project. **Any teaming or collaboration changes must be approved in advance by the Professor.**

Whether the project is individual or team-based, **the product baseline should be built frequently – at least once per day – and all of the automated tests run** to identify any breakage that needs to be added to the Sprint Backlog of tasks, prioritized, and addressed. **Build as small a set of functionality as can be tested, compile it, and test it.** Note any bugs found in the Sprint Backlog (task list) for prioritization and eventual resolution (though not necessarily before the end of that sprint). Then move on. **Do NOT write all of your code for the entire project and then ask us for help getting it to compile.** That's not how we teach software development, nor is it how professional developers operate in the real world. Time to grow up in your approach to developing software.

Code a little, test a lot, backup frequently.

In lieu of weekly sprint demos, students will submit "Evidence of Compliance" via Blackboard at the end of each sprint – each Tuesday at 8 am. You are expected to have completed at least the features indicated in the "Sprint # (by team size)" column that matches your team size. **Every team member must submit a deliverable each week to receive a grade.**

Final work products are due in Blackboard by Tuesday, December 3rd at 11:59 p.m. This is the next to last day of class. No extensions will be given. If you develop incrementally as we have taught you, then you **will** have a working program on that date - guaranteed. Deliver it, enjoy the Guest / TA lectures and project demos (hopefully being selected as a finalist to perform a demo yourself), and then prepare for finals.

Deliverables

Each student or team must submit the code for the project, representing that sprint's deliverable-

ready milestones, **must be delivered to Blackboard *every week*.** {50 points per sprint}

Each student or team is <u>required</u> to maintain and work from the Product Backlog and (for each sprint) the Sprint Backlog, using the Scrum spreadsheet provided. Each Sprint Backlog item (task) must be implemented by the single team member with whose initials it is associated. This will enable the graders to estimate the portion of work contributed by each team member. **Actual grades will be adjusted based on the actual work accomplished by each team member** – team members who contribute little will receive a *significantly* receive lower score than those who contribute much. **An updated copy of the Scrum spreadsheet must be delivered to Blackboard *every week*.** {30 points per sprint}

Each student or team is <u>required</u> to maintain and work from a UML design, updated as the project progresses. These will include *at a minimum* a class diagram and use case diagram, and eventually a state diagram when the associated feature is implemented. **Each student or team will deliver an updated set of UML diagrams in Umbrello format representing progress toward the final architecture and design *every week*.** These models must match your current code as well as represent your future planned work. You may use Umbrello in your Linux VM, or hosted in your native operating system (Windows, Mac OS X, BSD Unix, Linux, or whatever – file formats are cross-platform compatible.) {20 points per sprint}

The student or team will deliver a final proposal package to Blackboard that presents their capabilities in the best possible light by the homework due date. The proposal package will include:

1. Archive of the local git repository or link to the Github repository, showing all code commits in source code form; {50 points}
2. The Scrum spreadsheet representing the total project; {30 points}
3. Supporting UML diagrams including at a minimum a use case diagram, class diagram, and state diagram that accurately reflects the code, with additional consideration given for other useful supporting diagrams; {20 points}
4. (End of Sprint 6 only) A brief user manual (a few paragraphs); {10 points} and
5. (End of Sprint 6 only) Any sales material deemed to increase the probability of winning the contract (e.g., PowerPoint slides, PDF brochures, YouTube videos) {up to 25 points bonus}.

Each student or team should be prepared to provide a final Product Demo to the class, if selected and scheduled by the Professor after completion of the final sprint. The number of Product Demos, if any, is dependent on availability of class time. More info to follow. {Up to 50 points bonus}

Teaming

It is permissible but **<u>not</u>** required to work in teams of up to 4 students. Remember, if you decide to form a team, your grade will depend in part on the work ethic of your teammates. Choose wisely.

To form a team:

1. Talk with each invited team member to ensure they are enthused about the team. Be sure to discuss and reach agreement on how much additional work you want the team to accomplish for bonus / extreme bonus work.

2. Select a name for you team. It may be as creative as you like, within the obvious constraints of non-vulgarity, respectfulness, and professionalism.

3. When all are in agreement, EACH team member should email the Professor and TA, with cc: to each teach member. EACH email must include the name of the team in the subject, and the list of team members in the body.

4. The professor will Reply All to EACH email to acknowledge the team. This email authorizes the team. **If you do not receive a reply, your team is NOT AUTHORIZED** – contact the Professor for details ASAP.

5. **EACH TEAM MEMBER must submit the identical deliverable EVERY WEEK** to receive a grade. "My team member submitted it" will result in a grade of 0 for that week. This is a feature of Blackboard – we literally cannot enter a grade if you do not submit your own work.

Once formed, the team will continue for the duration of the project except for extreme circumstances. If you believe the team cannot continue as authorized, e.g., a team member withdraws from the class, contact the Professor ASAP.

Bonus and Extreme Bonus

Unlike previous homework, your opportunity for bonus credit is primarily rooted in *completing additional features within the 6 sprint project limit*. Thus, it is to your (or your team's) advantage to complete features ahead of the sprint schedule to allow time for additional features to be added by the end of the scheduled project.

Each additional feature implemented *in priority order* is worth bonus 75 points for an individual, 40 points each for a team of 2, 30 points each for a team of 3, and 25 points each for a team of 4 (as always, adjusted by each team member's contribution to that feature per the Sprint Backlog). If you wish to implement features out of priority order, be sure to receive written approval from the Product Owner (the Professor) *in advance*.

The Fine Print

As mentioned before, *the Product Backlog will change* during this project. Agile software development is all about effective response to change. You will have the opportunity to experience this over the next six weeks.

Robot images are used under license from Graphic Stock. Students may not under any circumstances use graphics provided along with this project for any other purpose, as such use would constitute copyright infringement. Remember, *we talked about this*!

To Get You Started...

Attached are some suggested artifacts to help you get started. These are not complete and are not sufficient for you to complete this project. Similarly, you may ignore any or all of these if you prefer, and create your own from scratch.

Robbie_Robot_Shop_Scrum_7.ods is the usual Scrum template for managing your implementation. If you still have questions on how to use this, see me during office hours.

Partial_UML_Class_Diagram.png is a partial, *incomplete* class diagram showing how various robot parts could be used to define robot models, and how robot models could be combined with customers and sales associates to produce orders. And over it all is the shop, running the business. This is a rough draft of a Model – you must supply the Controller and View.

Robbie_Robot_Shop_Sample_GUI_Design.pdf is a rough draft of a suggested menu layout for a GUI. Of course, you'll start by building a simple CLI, but when it's time for a GUI, here are a few ideas to get you started. You may want a tool bar in addition, for push-button software ease.

Robot_Images.zip are a collection of images of robots and robot parts. Note that these are licensed for use in this class, and my not under any circumstances be used for any other purpose whatsoever.

Update #1

About Change

If agile processes are about *rapid response to change*, you should expect change during this project – to the feature backlog, its priorities, requirements, etc. However, changes are constrained to occur at the *start* of the next sprint. Thus, what follows is intended as clarification rather than change.

Frequently Asked Questions

What's Team ID?

Team ID is just the name of your "company" that's responding to the proposal. Feel free to create a compelling team name, even if you're a team of one. "0x526F626F7473" is clever if unpronounceable. "Rossumovi Univerzální Roboti" is taken, sorry, although US Robots and Mechanical Men, Inc. might be free. Be creative, but if not, just use your student ID.

Do I need a Makefile?

You **always** need a Makefile for work submitted in this class. Please ensure that when the grader unzips your archive into an Ubuntu 16.04 directory and then types "make" at the bash command prompt, they get an executable that they can grade.

May I include a GUI in Sprint #1?

If you want to develop the model (aka MVC pattern) in parallel with the view, then the Product Owner is willing to allow your team to work on the later GUI features in earlier sprints. (This is the way Scrum works – the Team negotiates the order of feature implementation with the Product Owner. It's not based solely on the Product Owner priorities.)

The caveat is that the feature-to-sprint assignments won't be adjusted, so the later GUI features must be accomplished *in addition to* the agreed model features (e.g., sprint 1's product must still be able to create parts, even if via a GUI).

If you implement later features, ensure that your Scrum spreadsheet accurately records this!

Do I have to write regression tests for all of my code?

Your code should **always** have automated regression tests for all functionality, *except* the GUI user interface (for which it is exceptionally hard to automate regression testing). As always, running a test (or all of the tests) should produce NO output if no problem is detected, and the word FAIL if a problem is detected along with some diagnostics for each failure.

Does my class diagram only need to represent what I deliver in sprint #1?

Your class diagram should **always** represent the entire program, not just what you're delivering this sprint. It's strategic as much as tactical. So represent your GUI even if you're not delivering one this sprint. You **don't** need to represent test-only code, though you may if you like.

You are free to change your design, and thus your class diagram, each and every sprint. This also is normal in an agile process, which focuses on *rapid response to change*. Your code must conform to your design at the end of each sprint, of course.

Must I use inheritance? I'd rather just copy and paste code between my classes.

You are expected to use inheritance, polymorphism, and operator overloading on this project. That's why this class is named "Object Oriented Programming" - though even in structured programming, duplicating code was deeply frowned upon for what should be obvious reasons. However, you need **not** necessarily include them in the first week's sprint. Baby steps...

In general, any object that you print to the console should be via overloading of operator<<. If you can't find a use for operator overloading in your production program, make it part of your test code, because the graders will be specifically looking for this in an upcoming sprint (but *not* sprint #1).

Could you provide a Scrum spreadsheet example?

I've included 4 screenshots in Update #1 – a start-of-sprint-1 and end-of-sprint-1 view of both the Product Backlog and the Sprint 1 Backlog. Hopefully this will clarify how to manage Scrum with the spreadsheet. But don't just copy my tasks – please create your own based on your design!

Update #2

A Note on New Features and Requirements Changes

Remember that **the feature list is frozen during a sprint**. Because these changes were identified to you during Sprint #3, they can NOT be required for Sprint #3. They are therefore available for inclusion *starting in Sprint #4*.

Clarifications:

Screenshots: Starting with "open a GUI main window", please include a screenshot (in PNG format) of each window and each dialog in your GUI. This will help the graders ensure that they have built your program correctly.

VirtualBox Performance: If your VirtualBox performance is unacceptable, check your computer's BIOS. Changing from "Power Saver" to "High Performance" modes (if available) has made a lot of difference for some students.

Saving Umbrello Models: If your Umbrello installation still doesn't allow you to save models, first remember that you may use any recent version on any platform (including Windows and Mac) to construct your models. If you want to run Umbrello under Ubuntu 16.04 but save throws an error, try the solutions at https://bugs.launchpad.net/ubuntu/+source/umbrello/+bug/1585611, including:

- "**`sudo apt-get install kinit kio kio-extras kded5`**", or
- Downloading the prevous version from http://ftp.de.debian.org/debian/pool/main/u/umbrello/ and installing with "**`sudo dpkg -i *.deb`**".

Order Cost: The total cost of an Order must include taxes and shipping. Taxes are 8.25% of the price of the robot. Shipping cost is $15 per 100 pounds or 45 kg.

Stateful Orders: The feature "manage a robot order for a Customer via GUI" should include a simple state implementation to track the order's progress, as follows:

When placed, the order immediately enters the "Pending" state. The next states are entered via a "Begin" event (menu selection, button press, etc.) from user. The order then splits into two state paths - "Packaging" (which on a "Packaged" event transitions to "Shipping", and then on a "Shipped" event merges with the other state path), and "Billing" (which transitions on a "Billed" event to "Accepting Payment" state, and then on a "Paid" event merges with the other state path on a "Paid" event). When the state paths merge, the order enters the "Completed" state and then the state machine exits.

Before the order is shipped or paid, an alternate event is available labeled "Cancel". If this event occurs (the user clicks the "Cancel Order" button or menu item), then the state machine is interrupted, the final state is "Canceled", and the state machine terminates. Once the "Shipped" **or** "Paid" event is received, the option to cancel is void (i.e., the "Cancel" event is ignored for that order).

Note that the events would be menu items, buttons, or other input from the user. Thus, the user should have the following events available: "Begin", "Packaged", "Shipped", "Billed", "Paid", and "Cancel".

The states in which the order may be are “Pending”, “Packaging”, “Shipping”, “Billing”, “Accepting Payment”, “Completed” (final state), or “Canceled” (final state). If in “Packaging” or “Shipping”, it will also be in “Billing” or “Accepting Payment”. Not all events result in state transitions; it is worth bonus points if those that would not result in a state transition are disabled (“greyed out”) or otherwise unavailable for that order. You may use any technology to implement the state machine.

On the sprint in which you implement the “manage a robot order for a Customer via GUI”, or Sprint #4, whichever is later, you will need to add a UML State Diagram modeling the Order class’s state to your diagram set. (Note that an individual student is not required to implement this feature, but may as bonus work.)

New Requirement: Demonstrate Polymorphism

As we discussed in class, starting no later than Sprint #5, please ensure that your code demonstrates polymorphism in some way. Be prepared to promptly answer the grader’s email if they ask where to find it in your code.

New Feature: Calculate Robot Model Fields

This feature is prioritized as 2.5, and fits right under “define new robot models”. It is allocated to Sprint #4 for all sizes of teams. **Please add this to your Scrum Product Backlog no later than Sprint #4.**

Robot models should include additional fields calculated from values supplied by their constituent parts.

Total Weight: This is the simple sum of the weight of each of the parts. This should be visible to everyone in the UI.

Cost of Parts: This is the simple sum of the cost of each of the parts. This is distinct from the price assigned by the product manager. Note: If and when “enable basic login by role and customize the GUI” is implemented, the cost of parts should ONLY be visible to the product manager and pointy-haired boss. BTW, a good rule of thumb when pricing a readily assembled product in a competitive market is 3x cost of parts.

Power Limited: This Boolean is true only if the sum of the (max) power consumed by the Head, Arm(s), and Locomotor is greater than the sum of the max power provided by each installed battery.

Battery Life: This is an estimate of the typical life of the batteries before recharging is necessary. Calculate the average power consumption as 100% of the Head’s power, 40% of the (sum of the) Arm’s or Arms’ power, and 15% of the Locomotor’s power. Divide the sum of the batteries’ energy (x 1000 to convert to watt-hours) by the average power, and you’ll have the hours of expected battery life.

Example: The selected Head consumes 100 watts, each Arm consumes 240 watts, and the locomotor 500 watts. We have 3 batteries installed in the torso, each holding 1 kWh of energy. Average power consumption is 100 + 0.4*(240+240) + 0.15*500 = 367 watts. (3*1000 watt-hours) / 367 watts is about 8.2 hours of operation typical.

Max Speed: The locomotor has a maximum rated speed, but that assumes it is moving no more than 5x its own weight. Derate the maximum speed by the ratio of 5x the locomotor's weight to the total robot weight.

Example: The Locomotor weighs 300 pounds and has a top speed of 12 MPH. If the sum of the weight of each robot part (including the Locomotor) is 1420 pounds, maximum speed is still 12 MPH (because 300 x 5 is 1500). However, if the sum of the weight of each robot part is 1850 pounds, maximum speed is 12 MPH x ((300 x 5) / 1850) is 9.7 MPH.