Lecture Notes
October 13, 2020

“Assignment T0: Team Formation” due before class this Thursday, October 15
https://courseworks2.columbia.edu/courses/104335/assignments/486855

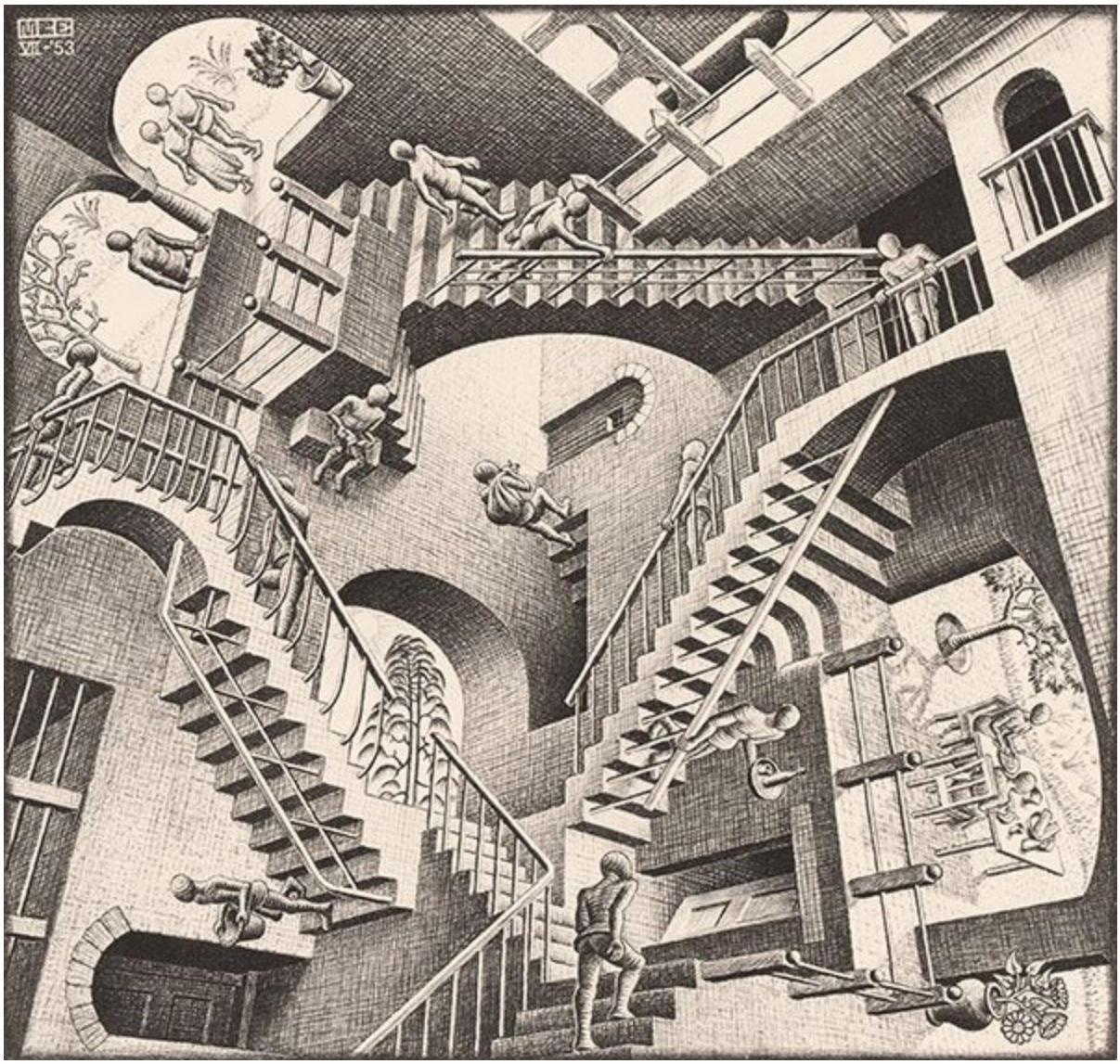
Try to form a team of *exactly* 4 members. Contact me or Shirish asap (via piazza, not email) if you’d like to form a team of 5 members. 2 or 3 members is not acceptable (except 3 is ok for CVN students teaming only with other CVN students)

If you have not found a team, or have a group with less than four people, please stay in zoom after class ends today to try to form teams.

Submit team members (full name and uni), team name, programming language(s) and platform(s), team github repository. If you want to use multiple languages and/or platforms, e.g., front end vs. back end, please explain. Everyone in the team must submit the same thing. If you still have not found a team, submit anyway.

Working in a team: software *design*

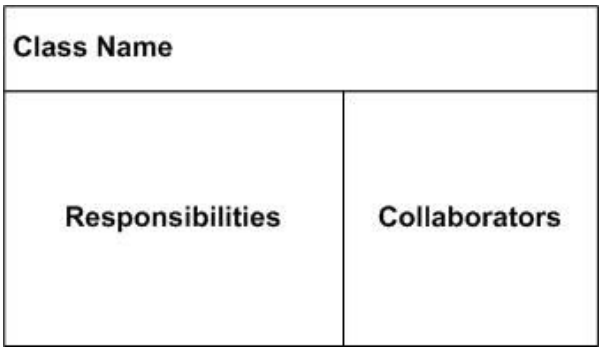
Here is one approach to design:



CRC is an easier, more practical approach to design for software engineering. Here is the original paper introducing CRC

CRC = Class Responsibilities Collaborators

Like user stories, CRC can be written on 3x5 (or 4x6) index cards, so often known as CRC cards



Each *class* is a thing, entity or object

Each *responsibility* is some action the entity needs to do

Collaborators are other classes the entity interacts with, communicates with, contains, knows about, or that otherwise help it perform one or more responsibilities

CRC focuses on the *purpose* of each entity rather than its processes, data flows and data stores (procedural design)

Here is a CRC card example for ATMs (automated teller machines)

How do we know what the classes (entities) are?

Start with user stories or, better, use cases

Find the *nouns* other than the user (user role) - classes should be the entities that “do” actions, not the actors who initiate actions

Convert plurals to singular, merge synonyms

Beware adjectives - may mean a whole new class, e.g., “car” vs. “toy car”

Beware hidden nouns, e.g., passive voice
“the thing is activated” = “SOMETHING activates the thing”

Some nouns may be attributes of entities, not themselves entities, e.g., “the radius of a circle” - here the circle is an entity and the radius is an attribute of the circle, it has no meaning as a standalone entity

How do we know what the responsibilities (actions) are?

Find the *verbs*

Say WHAT gets done, not HOW it gets done

Turn passive verbs into active verbs, as above
“the thing is activated” = “SOMETHING activates the thing”

The classes and responsibilities create a vocabulary for discussing the design

Where do the collaborators come from?

Find any relationships or associations among nouns (not necessarily symmetric, can be one-way)

How does CRC design work?

Develop a set of CRC cards for the current set of user stories (current iteration or upcoming release). Classes and Collaborators do not need to correspond to code classes, types or data structures, can be modules, packages, libraries, even files, databases, devices. CRC does not need an object-oriented programming language, C is fine

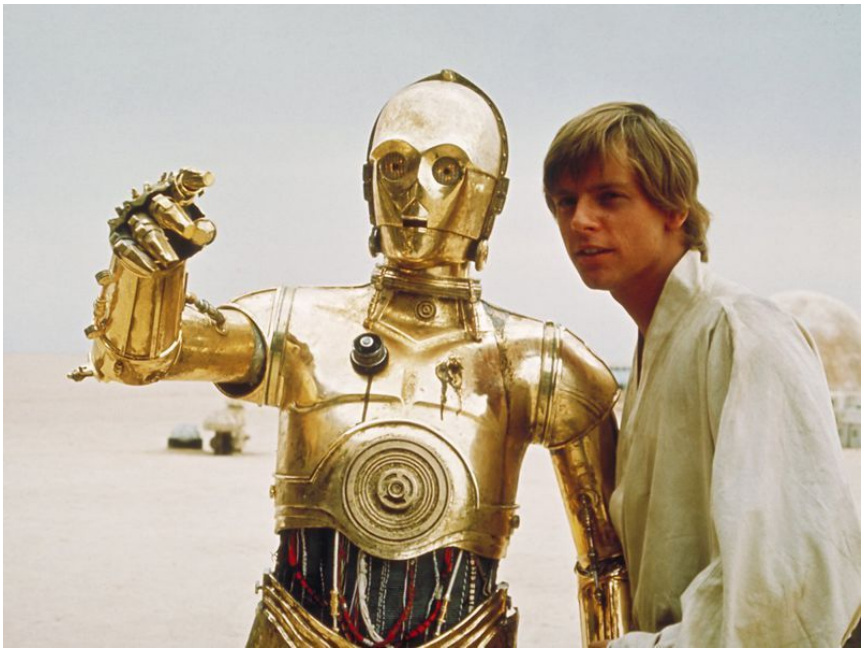
Walk through the scenarios of each use case and *animate* the CRC cards: move index cards on a table or sticky notes on a board. Make sure everything is a responsibility of some class, and figure out what each class needs from other classes to fulfill its responsibilities

When “the system” does something, that means some class in the system does it - there should not be any global master that does everything and/or is related to everything else. Give up global knowledge of control and rely on local knowledge of objects to accomplish their tasks. (This does not mean some classes cannot be singletons = there’s no more than one instance)

Then debug the CRC cards

- Remember classes are nouns that “do” operations, not the actors who initiate operations
- Split classes with too many responsibilities
- Combine classes with too few responsibilities
- Remove redundancy
- Remove classes with only CRUD operations (create, read, update, delete) that don’t “do” anything

Exercise: Put together some CRC cards for the laundry use case. Assume the housekeeper is a robot and you're designing software for this robot. The washing machine, dryer and iron are conventional machines that the robot uses. The robot has an ironing board for ironing, a table for folding, a garment rack for hanging, a disposal bin for throwing away, etc.



Let's try to make this work in zoom. One person in each breakout room could run this https://echeung.me/crcmaker/ and and share screen (it says you can share the link so everyone in group can edit in their own browser, but I could not get this to work). Or try something else…. Please come back in five minutes.

If you have not found a team, or have a group with less than four people, please stay in zoom after class ends today to try to form teams.

The first assessment will be posted this Friday October 16 at 12:01am, and is due by Monday October 19 at 11:59pm. You can spend as little or as much time during that period as you like. Submit early, submit often, but do not submit late.

The assessment will be based on lectures (watch the videos!) and the individual assignments. If you haven't already, you should follow the links embedded in the lecture notes and assignments.

The first and second assessments from the last time I taught this course are available in Courseworks Files https://courseworks2.columbia.edu/courses/104335/files/folder/from%202018/individual%20assessments
The first assessment references teams because the team project ran the entire semester last time, no individual project. The first assessment this semester will not reference teams or team projects, the second assessment will reference teams.

“Assignment T1: Preliminary Project Proposal” due October 27
https://courseworks2.columbia.edu/courses/104335/assignments/486922

Each team proposes their own project, within constraints:

- Must impose authenticated login.
- Must be demoable online (e.g., zoom or discord).
- Must store some application data persistently (database or key-value store).
- Must use some publicly available API beyond those that “come with” the platform. It does not need to be a REST API.

Describe Minimal Viable Product (MVP) both in prose - a few paragraphs - and via 3-5 “user stories”.

< label >: As a < type of user >, I want < some goal > so that < some reason >.

My conditions of satisfaction are < list of common cases and special cases that must work >.

Describe acceptance testing plan that covers these cases.

List the tech you plan to use. If different members of the team plan to use different tools, please explain.

Two sample team projects from a previous offering of this course are linked below. All submitted assignments as well as the code are included in each repository. These projects had three iterations because the team project ran the entire duration of the course, there was no individual project.

Code Phoenix - https://github.com/s4chin/coms_4156

Space Panthers - https://github.com/wixyFun/openSpace

If you have not found a team, or have a group with less than four people, please stay in zoom after class ends today to try to form teams.