**Defining a Process:**

**Software is designed to perform a process, and thus the skills you learn this month will aid you in communicating your software process to others.**

**What is a process?**

A process is defined as a series of actions or steps taken in order to achieve a particular end.

**Why is it important to define a process?**

It’s important to understand the benefits of defining a process in advance and completing up-front design to get approval and consensus before moving forward with development:

* Having peers review your project can lead to a more robust and error proof product
* A good software developer will analyze a project to identify gaps or flaws in the design before beginning coding
* Up-front design is the best way to show your client you are listening to them and getting their approval on the product they want
* Software follows same up-front design because when done incorrectly, it can be costly

**How to build/design a process**

**Every process has a start and an end (a pre- and post- condition)**

It’s up to the designer to determine and document the exact conditions for the process to start and end.

Example: Doing laundry could be defined as follows:   
*Pre-condition*: The clothes are in the hamper in laundry room and detergent is on the counter  
*Post-condition*: The clothes are clean, dried, and folded in the hamper.

Pre- and post- conditions should be determined before you start any work on a project so you and your client are in agreement on the scope. These are arbitrary statements and defining them is a key part of the process. These can be as detailed as you like.

**Use terminology your audience will understand**

Never assume people know what you’re thinking. Throughout your career you will be asked to communicate in terms that are understandable to your audience. You will need to differentiate and change your language context communicating with non-technical people versus your software peers.

**Determine the detail level**

* Based on your audience, can you use more technical terms and take a higher-level approach, or will it require something more detailed, in simpler terms?
* Again, much like pre- and post- conditions the level of detail is arbitrary and design is more about iteration. You can expand on each step to include more details. It’s useful to think of it as “OK, so how exactly do you do that?” once you have defined a step.
* As you design a process you should constantly ask yourself if the details are important. If details are important lock them down in your design for discussion and to create a contract of what needs to happen.

**Define the Actions**

* Determine all the steps that are needed to complete your process. Think of each Action as a step in a set of directions or instructions. You need every step accounted for in order to complete the process.
* Actions should be short and simple and must begin with an action verb. ie. *(Action) Display the final cost.*A program will execute the word immediately following your call out:*(Action) then****display****the final cost*would result in nothing as "then" is not a verb.
* You can have more than (1) Action in a sentence. *ie. (Action)****Remove****a slice of bread from the package and (Action)****place****on a plate. (Action)****Remove****a second slice of bread from the package and (Action)****place****next to the first slice.*is correct.Make sure you are calling out every verb in your sentence as its own Action.
* When dealing with multiples of an item, each one must be its own step:*(Action) Take out 2 pieces of bread and (Action) place on plate*would be incorrect as you haven't broken out each step, individually.  The program will only execute the first in the series. Correct: *(Action) Remove 1 slice of bread and (Action) place on plate.  (Action) Remove a second slice of bread and (Action) place on plate, next to the first slice.*
* Remember, if you use the word"and", this indicates that you have more than (1) Action and must call out as such. In the following example:*(Action) Open the cabinet, remove a plate and place on the table,*a program would only open the cabinet and either error, or skip ahead to the next correctly labeled Action.
* You cannot have negative Actions, ie.*(Action) Do not overfill*. You can not execute something that does not happen. To correct you might say, *(Action)****Pour****milk into coffee without overfilling*. If you find yourself compelled to say, *then (Action) do nothing,* simply delete this Action and continue with the process, as doing nothing is implied by moving on to the next action verb or Decision.
* For more examples of proper and improper Actions please see the *Week 1 Activity Example PDF*.

**Define the Decisions**

* Decisions are always "if statements" ie. *(Decision)****If you would like creamer in your coffee****, (Action) pour creamer into coffee.*
* If you have the word "or" this indicates that a Decision needs to be broken up to accommodate. If you're thinking: "Do you want to use milk or creamer?" you would write it as the following: *(Decision) If you'd like to use milk, (Action) pour milk into coffee, otherwise, (Action) pour creamer.*
* You can use the term "otherwise" for when you have a finite set of results. As above: milk or creamer. Do not use as a way to continue on with the remainder of the process. This is incorrect:*(Decision) If you'd like to use milk, (Action) pour milk, otherwise (Action) pick up the cup.*The Action to pick up the cup is going to be completed regardless of the Decision to use milk or creamer and is unrelated to the Decision.
* If you have a series of choices you must break each of these up into their own Decision followed by an Action. ie. *(Decision) If you want yellow paper, (Action) remove a piece of yellow paper. (Decision) If you want blue paper, (Action) remove a piece of blue paper. (Decision) If you want white paper, (Action) remove a piece of white paper. (Action) Place the paper on the table.*
* For examples of proper and improper Decisions please see the *Week 1 Activity Example PDF*.

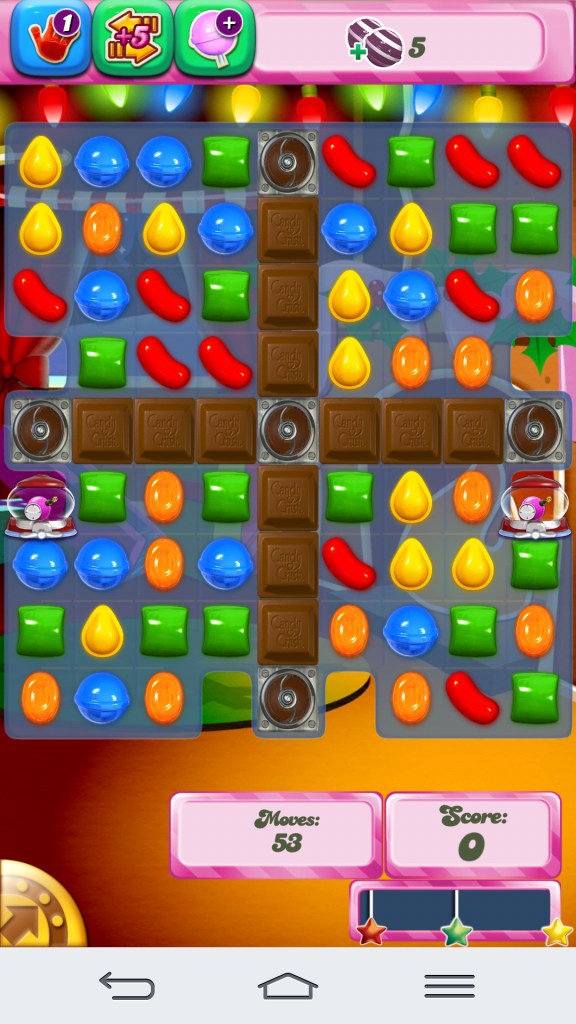
**Define the order of actions and decisions**

* Make sure your process is defined in a logical order to properly execute and satisfy your pre- and post- conditions.
* As the designer it is your job to assess the situation and communicate your ideas in the best manner possible.

**Logic & Decision Making:**

Any time you develop an app, it’s crucial to think about the logic and decision-making that your application will be executing when a user is interacting with it.

We’ve taken a look at the architecture of an application, how it displays content on different screens based on user input. Let’s take a closer look now at some of the decision processes an application must go through in order to execute interaction on a given screen.

Using the popular application, *Candy Crush Saga*TM as an example we will take a deeper look at some of the logic behind the game play screen.

Above we have our opening game screen once we’ve chosen a level. The goal is to execute the objective for any given level in a specified amount of moves and/or time. You’ll see we’re starting with 53 moves and a score of 0 to reach our objective at the top right of the screen.

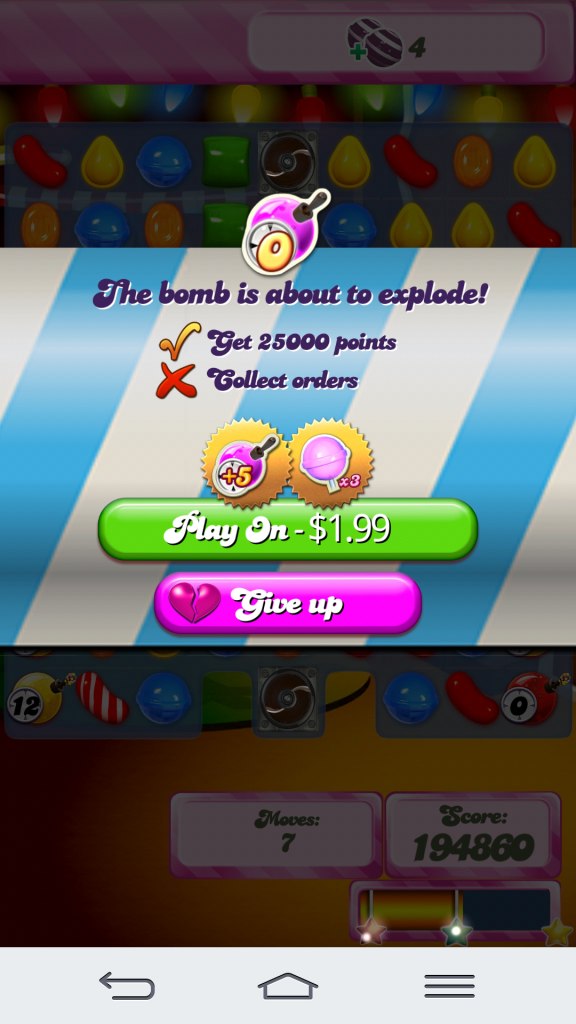


During game play the application has several decisions to make based on user interaction.

When pieces are moved:

* Do they make a match of 3 or more candies? If so, add points to the total score based on amount of candies involved, remove said candies, shift remaining candies, deduct (1) move and deduct (1) round on the bomb to the left.
* If they do not make a match of 3 or more candies, add zero points, deduct (1) round on the bomb. Retain all candies.
* Do they create a special piece such as a striped, wrapped candy or sprinkled candy? If so, remove appropriate pieces and replace with the special candy/ies, shift remaining candies, deduct (1) move, add points to the total, and deduct (1) round on the bomb.
* Do they satisfy one of the objective/s? If so, update objective at top and execute one of the above decision flows. If not, leave objective as is and execute one of the above decision flows.

Based on moves left, objective/s and/or number of rounds left on the bomb, the application must decide if the game will continue or end.



If game play continues we repeat the above decisions. In this example, we have counted the bomb down to zero rounds before our moves are (0) and before we meet the objectives. The game must decide to end the round and show a dialog box, asking the user if they would like to give up or continue playing for a fee.

It has continued counting down moves, added to the score as appropriate and updated the objectives at the top based on prior game play.

It’s important to note that software processes are in fact very simple: the complexity comes from combining lots of simple actions and decisions.

**Activity Diagrams**

An Activity Diagram is the UML diagram that provides a clear and logical way for you to display "how" your software application is going to work:

* Symbols used are intuitive, for easy explanation.
* A map of your application to follow control flow, show sequence of events and decisions occurring during execution.

#### Activity Diagram Symbols



**Initial Node**

Every Activity Diagram shall have one initial node, preferably at the top of the diagram. This is the entry point of your process.



**Final Node**

Every Activity Diagram shall have at least one final node, preferably at the right or bottom of the diagram in keeping with the top down, left to right theme.



**Action**

* The Action symbol contains the action being performed.
* Actions can be put in sequence connected by control flows to show the order of operations.
* The Action should be short and simple, including a verb.
* You are encouraged to add as many actions in sequence as you need to tell your story.



**Control Flow**

The control flow shows the direction of the process.



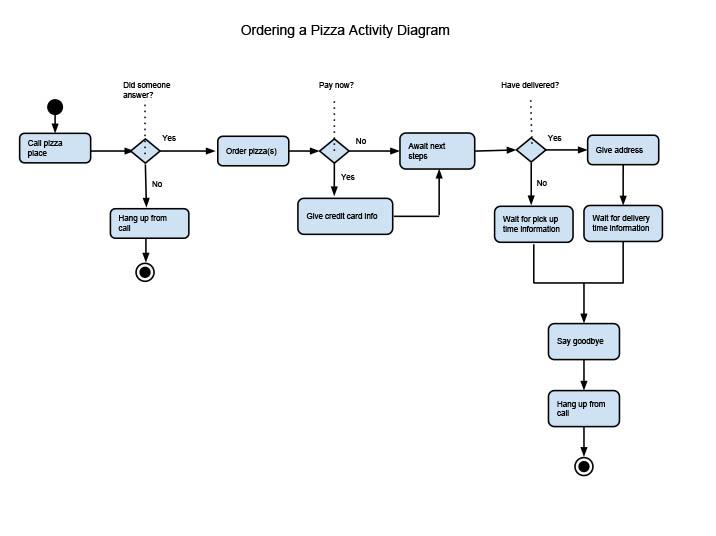
**Decision**

* The decision symbol shows a split in actions based on a condition
* Outward control flows must have a statement of the condition that brings the process to that control flow.
* The condition should clearly state the decision.
* Think of your decisions as a yes or no question to easily split your actions.
* These are your "if statements" in your written process. Think of this as: "if yes" or "if no" responses, once you form a question that encompasses the decision.

#### Activity Diagram Example

**Non-System Process**

First we'll be studying a non-system process such as ordering a pizza.



**Note**: "*Await next steps*" does not imply that we're waiting for the next step in the process, but rather literally awaiting next steps as instructed by the pizza parlor.

Take note of the level of detail we've chosen to display. What are some other details we could have included in this diagram? Perhaps decisions or actions pertaining to:

* Whether or not we make our own or choose a specialty
* If we get pizza toppings
* If we choose to ask about specials running

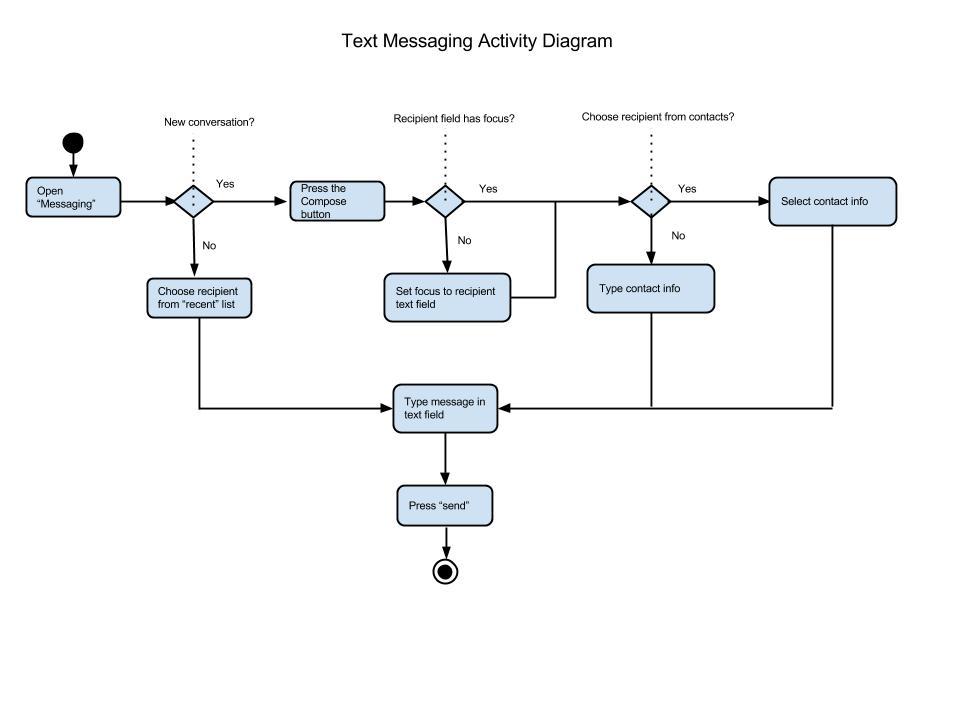
These diagrams can be extremely detailed, or a higher level depending on what you're trying to accomplish. What's important is that all the steps shown are necessary to order a pizza. Keep this in mind when creating activity diagrams. What is a minimum level of detail required to execute the process and are there times when much more detail is required?

Below I have a new written process, sending a text message. Take a look at the process and then continue to find the Activity Diagram generated from the textual process:

**Pre-condition**: Mobile device is turned on and navigated to an appropriate Text Messaging button to launch application.

**Post-condition**: Text message is sent.

(**Action 1**) Open Text Messaging application. (**Decision 1**) If this is not a new conversation, (**Action 2**) choose recipient from recent messages list and (**Action 3**) proceed to Action 8. (**Decision 2**) If this is a new conversation, (**Action 4)** press the appropriate Compose New Message button. (**Decision 3**) If the recipient field does not have focus, (**Action 5**) set focus to the recipient text field. (**Decision 4**) If you are choosing a recipient from contacts, (**Action 6**) select appropriate contact info. (**Decision 5**) If you are not choosing a recipient from contacts, (**Action 7**) type contact info into recipient text field. (**Action 8**) Type desired message into appropriate outgoing message text field. (**Action 9**) Press send.



Take a look at the above activity diagram and take some time to think about what other details or branches of the process could be elaborated upon for even more in depth decision-making.

**Note**: You, as the designer can present the actions in the order you believe is best. Having presented the design on paper, you can discuss changes to that order with your client.

**Non-Linear Activity Diagrams**

An Activity Diagram is the UML diagram that provides a clear and logical way for you to display "how" your software application is going to work:

* Symbols used are intuitive, for easy explanation.
* A map of your application to follow control flow, show sequence of events and decisions occurring during execution.

#### Activity Diagram Symbols

We'll be introducing a new diagram symbol in this week's activity and assignment.



**Process**

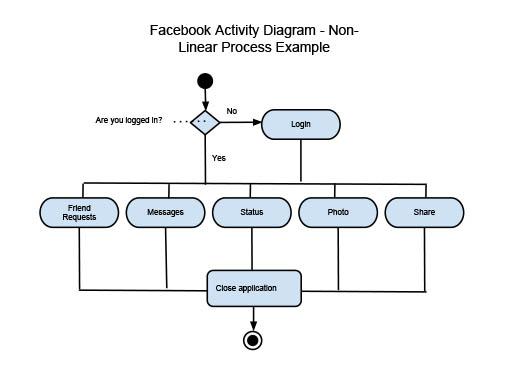
As applications are made up of nested processes - both linear and non-linear - this symbol, along with a descriptive name represents the start of each of those processes available to you in the application.

#### Activity Diagrams

Let's dive into an Activity Diagram for the Facebook app, which we've briefly outlined in our previous activity. I've chosen (5) possible processes that are available to us in Facebook for the purpose of this example.

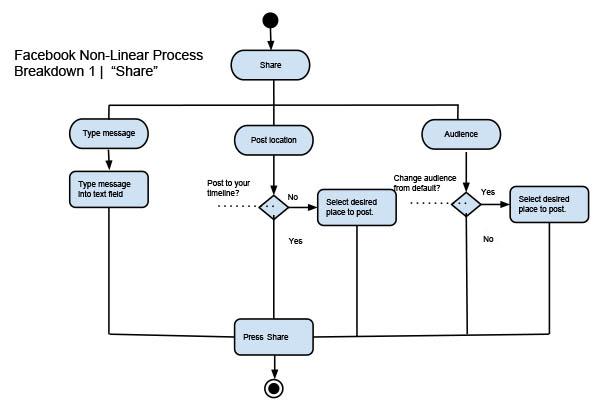
**Pre-condition**: Facebook application is open.

**Post-condition**: Desired process/es are complete and the application closed.

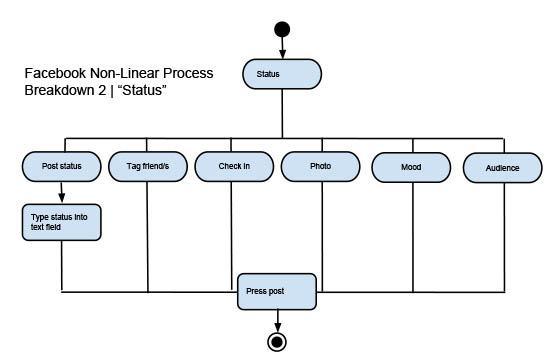


You'll see above that these are (5) different options we have within the Facebook app. We've used the process symbol to illustrate, as each of these go into their own non-linear processes. Now we'll break down (2) of those processes into their own charts as you will be required to do in your Week 4 assignment.

Below is the 1st process we'll pull out of our over arching non-linear Facebook process, Share.



Above is what it would look like to diagram all (3) of the linear processes within "Share". Each end with the "Share" button being pressed but are not dependent on each other in order to execute the desired outcome.

Below is a breakdown of process 2 - Status Update.  
  


As you can see, this time we've chosen (1) linear process within the non-linear Status process to diagram. This process requires (2) steps and is independent of the other available options to the user.