j.Midterm quiz

***Basic stages of development***

**Requirement:** Gathering and analysis, have meeting with manager, stock holder and user to make sure the what is the requirements, such as who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system. After requirement gathering, analysis for their validity and the possibility for include them in the development going to start.

**Design**: In this stage, the system and software design are prepared from the requirement stage. System design helps in specifying hardware and system design also it help defining overall system architecture. System design specifications serve as input for the next phase of the model. In this stage, the tester comes up with the test strategy, where they mention what to test, how to test.

**Implementation/Coding:** After receiving system design documents, divide the work into module and start the actual coding. Since, in this stage the code is produced so it is the main focus for the developer. This is the longest stage of the software development life cycle.

**Testing:** After the code is developed it’s tested against the requirements to make sure that product is actually running. In this stage, all types of functional testing like unit testing, integration testing, system testing, acceptance testing are done as well as non-functional testing are also done.

**Deployment:** After successful testing the product is delivered / deployed to the customer for their use.

***Difference between Waterfall and interactive development***

**Waterfall (Big bang):** Form the requirement to design, from the design to code, from the code to test, from the test to deployment, all the stage need to try to do it best. Require every step to be organized and perfect.

**Advantage:** is providing a check point of each phase. When finish one phase, you just need to focus the next phase.

**Disadvantage:** The division of each stage is completely fixed, produce large number of documents between stages, which greatly increases the workload. Linear development model, user can only see the result until the very end, increase the development risk. Also, doesn’t adapt to changes in user needs.

**role of customer:** In the requirement phase, give the requirement to the developer. And In the deployment, end of the development receives the software to use.

------------------------------------------------------------------------------------------------------------------

**Interactive development:** Dose not require every phase to be perfect, even there are still lack of something. Focus on the main part first, and use least time to make a unperfect product for the customer, and receive the feedback from the user and fix the product.

**Advantage:** Reduce the risk, make a unperfect software early for determine the risk to have enough time to fix it. Improve the speed of the development, developer know the focus of the problem, their work will be more efficient. More adapt to the changes in user needs.

**Disadvantage:** The requirements for developer are high, need a high-quality project manager and a high-tech development team to adapt he early development of project changes.

***User stories:***

A user achieves something he went through a system (such as buy a coke), that is user story.

**What are User stories?**

Let’s assumed the producer of the vending is the client, and we develop a software for the vending machine. Client said “User enter a coin; machine show how many coins have user been inserted. When inserted coin is enough for buy certain drink, the light of that drink will on. If user press the button, vending machine will give the drink to the user, then give user changes.” From this story, we need to record it in a different format

------------------------------------------------------------------------------------------------------------------

Name: Vending machine sell drink

Story:

1. User inserted money.

2. Vending machine display the amount of money inserted.

3. Drink light button on when the money is enough for it.

4. User press the button that have light on.

5. Vending machine sell a drink.

6. Vending machine give user changes.

Help client and developer communicate. **A user story includes 3C**.

**1. A written story description used to make plans and as a reminder. (CARD)**

**2. Conversation about the story, used to materialized story’s detail. (CONVERSATION)**

**3. Test, used to express and document story details and can be used to determine when is completed. (Confirmation)**

**-------------------------------------------------------------------------------------------------------------------**

**What is the good user story? (INVEST):**

1. Independent 2. Negotiable 3. Valuable to users or customers 4 Estimable 5 Small 6 Testable.

**Guideline:**

1.Make the size of the story let the user feel that they can go for a coffee break after use.

2.Don’t let story involve user interface too early.

3.Sotry for individual user.

4.Including the user role when actually writing the story.

5.Let user writing the story, not developer.

***Estimation:***

Assume the size of the project, we always use point for this estimation. Planning poker is a common way, the point on the poker is 0,1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?.

The entire team have discussed for the project. Every member give card for estimate, then pick the outlier, let they explain their reason for why need that long or short, sometimes other members may forgot something and underestimate the difficultly of a project.

**What’s the largest point and smallest point?**

The one we discuss in the class the largest one is 13, smallest one is 1. If any estimation is larger than 13, develop team need to split the big chunk into smaller pieces.

**What’s the advantage?**

Help the team estimate the time and every developer can communicate with one another. This is preparing for Iteration planning meeting.

***Planning***

Iteration: Iteration will continue running to get the correct answer. For example, pick someone’s favorite music. Just keep picking it until bingo.

**Utopian vs Real-world days:** Developer may underestimate the difficult of the project, and didn’t plan enough time for it, real world days will estimate with more reality thinking, include the velocity of the work.

***Big Board:*** Reference as Kanban, invented by Toyota as a great way to analysis the work status. For software development, it can make four part on the board: waiting, working, stuck, done. A board let improve the efficient and develop team can read the current status.

***Velocity:*** Velocity reference the efficient of the work. Recommend rate is 0.7, change in velocity means the time for the development going to increase or decrease. If increase the velocity it will probably bring some bad result, because tired will effect productivity. Study shows that most developer have three hours of striking productivity, we shouldn’t push the team do too much work, more time doesn’t make it better.

***Design:***

**Good enough design:** Design the program is good enough for use, to make sure finish the work and left some space for improvement, for example, the video games always came with bugs when it just release, but the update will add more detail and make it better.

**DRY (Don’t repeat yourself):** Reduce the repetitiveness code. Every part in the system should have a clear authority representative.

**SRP:** Unix/Linux is a perfect example, each program handle one individual work. Easier to understand, easier to maintain.

**SOLID:** It include SRP, OCP, LSP, ISP, DIP

**OCP:** Open-closed principle, extension but modification. Example: A store provide multiple payment method (Alipay, Wechatpay). Less testing, less code to review.

public class PayHandler {

public Result<T> pay(Param param) {

if(param.getType() == "ALIPAY") {

// use alipay

...

} else if(param.getType() == "WeChatPay") {

// use wechatpay

...

}

}

}

But if we want to add new payment method or edit a method, we need to edit the PayHandler class, it may affect the code of now. Bettwe way is abstract different method, like below.

public class PayHandler {

private Map<String, PayProcessor> processors;

public Result<T> pay(Param param) {

PayProcessor payProcessor = processors.get(param.getType());

// 异常处理略

return payProcessor.handle(param); }

}

interface PayProcessor {

Result<T> handle(Param param);

}

public class AlipayProcessor implements PayProcessor {

...

}

public class WeChatPayProcessor implements PayProcessor {

...

}

In this case, we just need to add new class for new payment, it wouldn’t affect the payhandler.

**LSP :** Liskov substitution principle, an object can be replaced with a subclass instance anywhere it appears, without causing program errors. Example: Bird class can fly, duck is inheritance of bird, so it can fly, but chicken is inheritance of bird and they can’t fly.

Square is a subclass of rectangle, and the square’s length == width. Different than the others. So, set length == width in the square function. But square still a subclass of rectangle.

Knowledge about base-class appl to sub-class, easier to understand, easier to maintain.

**ISP (Interface segregation principle):** Client shouldn’t be forced to rely on methods it doesn’t use. Interface should split to smaller and more specific that can help refactor and change.

Example: Still use the payment example, set Alipay suggest pay and refund. Wechatpay only accept pay. More cohesive, lower coupling. Easier to understand and maintain.

interface PayChannel {

void charge();

void refund();

} class AlipayChannel implements PayChannel {

public void charge() {

...

} public void refund()

{ ... }

} class WeChatChannel implements payChannel {

public void charge() { ... }

public void refund() {

// 没有任何代码 } }

In the following code, the wechat pay don’t include the refund. However, due to the implementation of PayChannel, the refund() as an empty method. Then in the call, this method can be called, but done nothing. There is way to fix it. It’s to split PayChannel into PayableChannel and RefundableChannel.

**DIP(Dependency Inversion Principle):** High level module shouldn’t rely on low level modules, they should both rely on abstraction. Abstraction shouldn’t reply on implementation, implementation should rely on abstraction. DIP is the reverse of it. Lower coupling, reuse, easier to test, understand and maintain. In fact, DIP is the method to implement Open-closed principle.

Example: In this case, payHandler rely on two low level processor.

public class PayHandler {

public Result<T> pay(Param param) {

if(param.getType() == "ALIPAY") {

AlipayProcessor processor = new AlipayProcessor();

processor.hander(param);

...

} else if(param.getType() == "WeChatPay") {

WeChatPayProcessor processor = new WeChatPayProcessor();

processor.hander(param); ...

}

}

}

public class AlipayProcessor { ... }

public class WeChatPayProcessor { ... }

***UML*** :

Association: A has access to some B, but B have nor access to A.

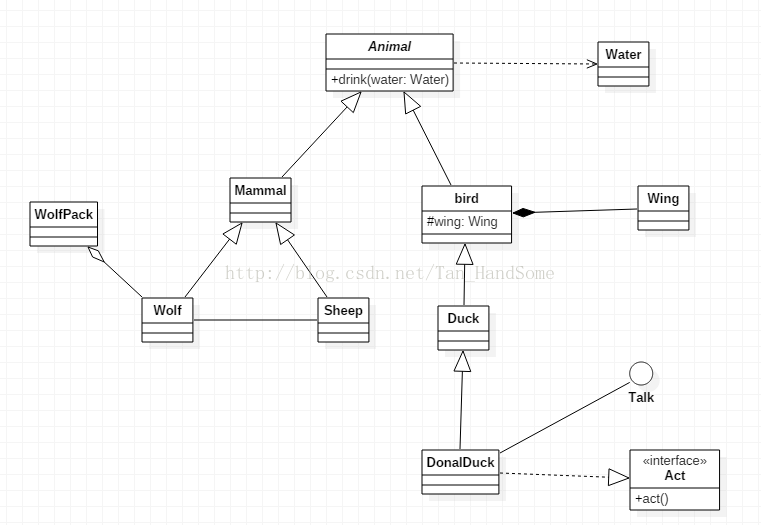
Generalization: A is a kind of B.

Realization: B is implanted by A, like act is realization to duck.

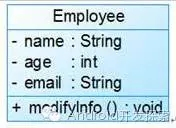
Dependency: A depending to B, like animal rely on water.

Aggregation: A is part of B, no include all B but same kind. But A can be existing by itself. Like wolf pack is aggregation of wolf.

Composition: A is a smaller part of an individual B, like a wing is composition of Duck. The wing can’t be existing, unless its pair with duck.



Wolf pack is a group of wolf (Aggregation), both wolf and sheep are belonging to the class mammal (Generalization), Animal need water (Dependency), Wing is a part of bird’s body(composition). Act is implementation of DonalDuck (Realization).

 +: public, -: private, #: protected.