**Workshop seminar 5**

**Comp201 Software Engineering**

**Software Engineering Software Design**

**Review Questions**

These questions are to help you review your understanding. Please answer the questions to the best of your ability, refer to the power point slides and the lectures to help you understand.

1. Given we have a large project for an e-commerce site, with over 600 use cases and will be split into 6 sub-systems, payment processing, order processing, complaints and returns, marketing and search engine (including smart recommendation engine), security management

We have a team of 60 developers assigned to us.

In this circumstance, why is a detailed software design important?

**We have 60 developers, what tasks will they be doing and in what order.**

**So we need to allocate our developers into teams for each sub-system.**

**We may also need a UI team who will working on the web site services, this is often a large part of the work.**

**When we are allocating people within the team, what software is each member of the team meant to produce, so in Java, what classes?  
So think of the payment system, given we have to develop interfaces to the following payment systems;  
Worldpay, Paypal, Sage Pay, Amazon Pay, Apple Pay  
For each one, how much work is required, what classes are needed, can be re-use software from vendor.  
We can then start splitting out developers into different teams for different sub-systems.  
For each sub-system we can also decide how to test it, so we can develop a test specification, how do we verify customers details, how do we manage storing customers details, do we store card details directly, do use a tokenization service, so we don’t have to store customers card details at our site?**

**What might happen if we don’t have a proper software design?**

**The software might be difficult to integrate.  
The software may well be poorly tested.  
The software may be hard to re-use on new systems.**

1. What is the benefit of using the following Java keywords in terms of software characteristics, see slides in lecture 14;

private

package

class

extends

**private established encapsulation and reduced coupling.**

**You can use getters and setter to establish the use of the class.**

**Example code, Security encapsulation**

**class UserLogin {**

**private String password;**

**public boolean checkPassword(String password) {  
 return(this.password.equals(password);**

**}**

**public void setUsername(String username) {  
 // TO DO… set password from username loaded from database**

**// String sql=”SQL select password from user where username=’”+username+”’;”;**

**}**

**}**

**// Notice in this case the user of the class, cannot access the password, there is no getter for // it, the user of the class cannot get access to the password, so is more secure**

**public class Postcode {**

**private String regexPC = "^[A-Z]{1,2}[0-9R][0-9A-Z]? [0-9][ABD-HJLNP-UW-Z]{2}$";**

**private Pattern patternPC = Pattern.*compile*(regexPC);**

**private void validate(String code) {**

**Matcher matcher = patternPC.matcher(code);**

**if (!matcher.matches()) {**

**throw new IllegalArgumentException("Bad Postcode format");**

**}**

**}**

**public Postcode(String code) {**

**validate(code);**

**this.code = code;**

**}**

**private String code;**

**public String getCode() {**

**return code;**

**}**

**}**

**// You can protect the class, by providing validation on the incoming data**

**package**

**You can move a number of classes together into one package, if you the classes in the package, can have package private class.**

**package security;**

**class Login {**

**}**

**package security;**

**class Encryptor {**

**}**

**package security;**

**public SecurityManager {**

**}  
In this case, outside the package, you can only access SecurityManager, this is an example of the Façade structure, look at lecture 14, slide 11.**

**This dramatically reduces coupling, outside the package, other classes, can only access SecurityManager.**

**class**

**Encapsulates the data, with the code.**

**extends can be used to provide code reuse, this of a banking application.**

**class Person {**

**}**

class Customer extends Person {

}

class BankCustomer extends Customer {

}

You can now re-use, Customer on any future project that deals with customers and re-use Person on any new project which needs to process people.

So for a medical system, you could have

class Patient extends Person {

}