Software Engineering: TMA 03

MATTHEW MASON
C6122243

Table of Contents

Question 1

a.

i.

Three groups of stakeholders that are identifiable for the BillyDoo Software System consist of the following:

1. Customers: One concern that may influence the architecture of the design for this group is the quality attribute, usability. Customers would be concerned with how easily they can navigate the website, search for specific free or paid events in a location of their choice, and reserve or buy tickets.

2. Advertisers: One concern that might influence the architecture of the design for this group is the software's core feature - The ability to sell or give away tickets via BillyDoo's website for events they are organising and how effectively the system allows them to set up, manage, and advertise these events.

3. Venue Management: The main concern from this group that will affect the architecture is safety. These might include a need for verification of approved events at each venue to prevent fraud, accuracy of details of the venue such as maximum capacity for health and safety reasons or financial concerns like commission paid on events or how non-refundable deposits are managed in the event of cancellations.

ii.

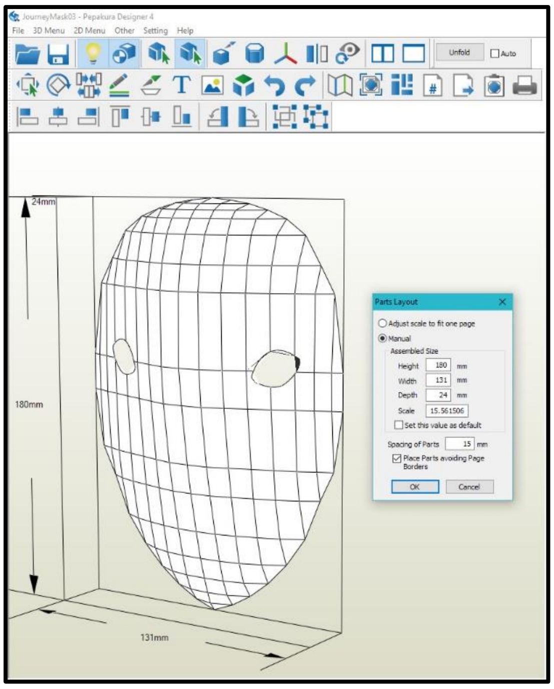
There are a number of architectural styles that might be involved in this particular scenario.

1. Client-Server: Clicking on the link in the email opens a web browser which in turn, sends a request to the server over a network for the specific page.
2. Layered: The Client-Server style is an example of a layered architecture with only two layers. The client uses services provided by the layer immediately below, in this case the server, to find and respond to the client with the requested webpage.
3. Notification: users subscribe to receive updates when a new event is published based on event filters and these updated are sent directly to each user's email.
4. Data Flow (Pipes and Filters): When an event is created, before a notification can be sent to users, the event must meet a specific set of criteria for each user. Filters can be used to determine if the set of criteria for an event matches a user's preferences and only then will the user be notified.

b.

An application that's a good example of the model-view-controller (MVC) pattern is Pepakura. It allows users to unfold 3D data into flat patterns which can be edited, printed and reassembled into physical 3D models.

As an example, the following figures show a 3D model of a mask which is made up of vertices, faces, and edges. This is the view the user sees and any changes to the data's state is propagated to the view such as when using the "unfold" button which in this case acts as a controller, processing the underlying information of the 3D model and converting it to a 2D representation.


JourneyMask03 - Pepakura Designer 4
File 3D Menu 2D Menu Other Setting Help
Unfold
Auto
24mm
180mm
131mm

Parts Layout

( ) Adjust scale to fit one page
(●) Manual

Assembled Size
Height 180 mm
Width 131 mm
Depth 24 mm
Scale 15.561506
☐ Set this value as default

Spacing of Parts 15 mm
☑ Place Parts avoiding Page Borders

OK Cancel

Figure 1: A 3D Model of a mask in Pepakura

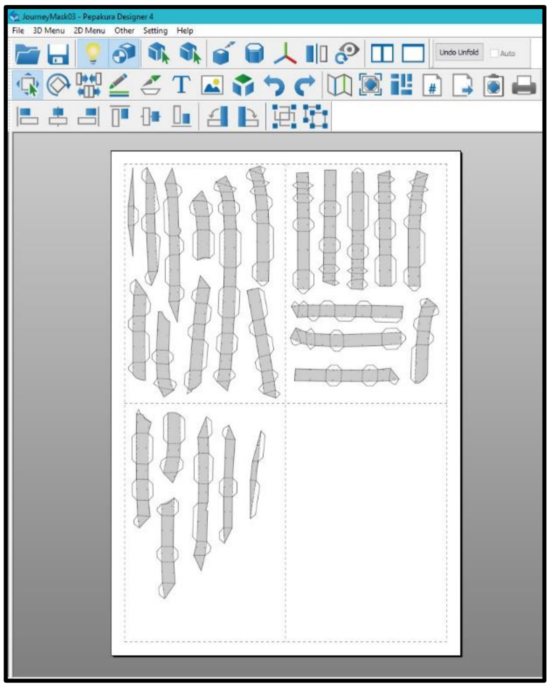

JourneyMask03 - Pepakura Designer 4
File 3D Menu 2D Menu Other Setting Help
Undo Unfold
Auto

Figure 2: Unfolded 2D version of the mask from figure 1

The advantages of the MVC pattern using Pepakura as an example, is the view is only concerned with displaying the 3D model, not how to change the data, meaning the domain logic can be used with different interfaces such as the 2D and 3D windows and can be tested separately from the interface logic.

Question 2

a.

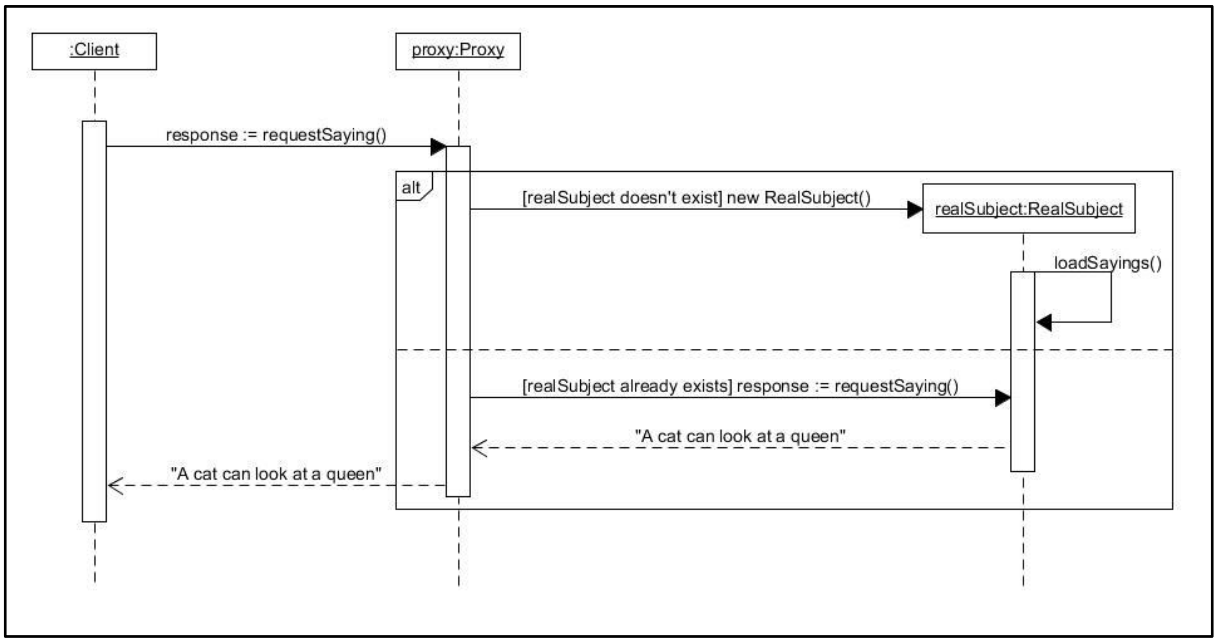

:Client
proxy:Proxy
response := requestSaying()
alt
[realSubject doesn't exist] new RealSubject()
realSubject:RealSubject
loadSayings()
[realSubject already exists] response := requestSaying()
"A cat can look at a queen"
"A cat can look at a queen"

Figure 3: Sequence diagram illustrating the interaction between the client, proxy and real subject.

b.

i.

One way in which both the proxy and adapter patterns are the same is that they are both structural patterns that deal with the relationships among classes and objects, using composition to control access to another object.

In the proxy pattern, both the proxy and real subject objects realise a common interface. The proxy objects primary purpose is to control access to the real subject, whereas in the adapter pattern, only the adapter class realises the interface and provides the client with operations that forward request to an object with an incompatible interface.

ii.

The degree of coupling in the proxy pattern would be loose. As both classes realise the same interface, the signatures that have been defined can be implemented independently in each class that realises the interface. This makes the proxy class transparent to the client so when it receives a request it can perform some additional task if necessary and forward the request on to the real subject. It makes each class more flexible with the ability to vary the classes independently.

iii.

A factory pattern can be used in conjunction with the proxy pattern. Where a factory or product created by a factory realises an interface, a virtual proxy, which uses a reference to the real subject, can be used to control access to either the creation of objects through the factory, or use of the product created by the factory.

Question 3

a.

i.

The main features of monolithic applications that contrast them from microservices are that all the capabilities and services provided are placed within a single application whereas the microservice approach takes each capability and puts them in a network of separate communicating processes. In terms of distribution, a monolith application scales by copying the application on to multiple workstations as opposed to microservices where flexibility allows different services to be places on different machines.

ii.

In a microservice style architecture, the individual services are components themselves that communicate using web service calls or messaging. These separate services afford independent run time processing and can be independently replaced or upgraded as opposed to monolith components which could consist of language dependant libraries making upgrading or replacement difficult.

iii.

Microservice development teams normally organise themselves around the business capabilities with each team focusing on communication all the way through the stack and the end users themselves. As an example, development teams could be separated into managing services such as orders or shipping for an online shopping experience. This is opposed to monolith applications where development teams are focused around the technology, such as Databases, User Interfaces, Serves, etc…

iv.

Microservice architectures organise and manage data via decentralisation. This means every service should be responsible for its own data, its own persistence and each service can only communicate with another services data store through its API. This has the advantage that the choice of data persistence and data store used is individually determined by each service, for one it might make sense to use a relational database, for another it might want to use NoSQL. It also removes the need for integrating services through a single database.

v.

Developing and deploying an application consisting of multiple remote services necessitates an automated way of distribution. This requires a need to design for failure and puts a lot of pressure on having a good monitoring system in place as remote services will fail, and so, when things go wrong, its easy to spot that an error has occurred and to be able to use the monitoring tools to debug the issue.

vi.

The advantages of a microservice style of architecture are:
1. The ability to deploy and upgrade services independently and quickly as opposed to a monolith application architecture where everything would need to be upgraded at the same time.
2. They give a greater degree of availability, for example if a recommendation service goes down while shopping, you can still run your shopping cart service. Whereas is a monolith application goes down, all services may stop running.
3. It preserves modularity and communication is purely through network interfaces so reduces the

temptation for coupling.
4. They allow for use on multiple platforms and languages across services providing flexibility.

The Disadvantages of a microservice style of architecture are:
1. It loses simplicity and not understanding module boundaries can lead to poor design. In comparison, a monolith application has a relatively simple and familiar approach to development.
2. Maintaining consistency becomes difficult across remote services.
3. Refactoring also becomes difficult as opposed to a monolith application where an object can be moved from one module to another providing there is a well-defined boundary between them.

vii.

The prerequisites that are necessary before adopting a microservice approach include being able to rapidly provision new machines, have at the very least, basic monitoring, make sure that services can be automatically and rapidly deployed and make sure operation and applications groups are communicating and working together.

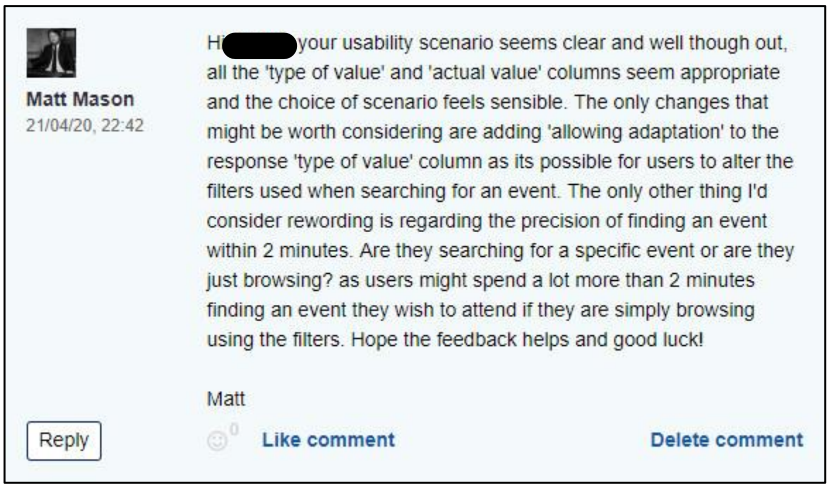
b.

An advertiser wishes to publicize their upcoming events using the BillyDoo software system. To do this, the website requires anyone wishing to advertise their events, free or paid, to complete the registration process. Part of this process is to select a unique username. When the user beings typing their username, providing it adheres to naming conventions specified by the website (such as containing a certain number of character, capital letters, etc…) results of whether the name is acceptable should be returned to the user within 1 second of any keyboard inactivity.

Part of Scenario	Type of Value	Actual Value
Source	External	Web browser, internet user
Stimulus	Sporadic	Selecting username
Artefact	System	BillyDoo system
Environment	Normal	Browser and internet connection working normally
Response	Process stimuli	Return confirmation that username is acceptable or not
Response Measure	latency	Confirmation returned with 1 second of keyboard inactivity

c.

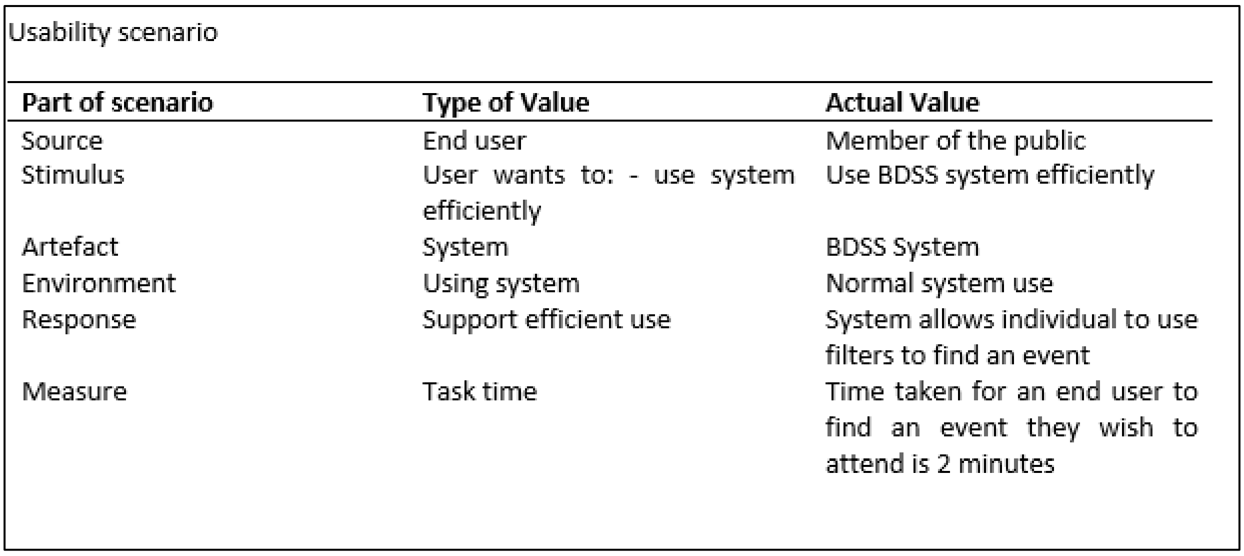
My comments to [redacted]


Matt Mason
21/04/20, 22:42

Hi [redacted] your usability scenario seems clear and well though out, all the 'type of value' and 'actual value' columns seem appropriate and the choice of scenario feels sensible. The only changes that might be worth considering are adding 'allowing adaptation' to the response 'type of value' column as its possible for users to alter the filters used when searching for an event. The only other thing I'd consider rewording is regarding the precision of finding an event within 2 minutes. Are they searching for a specific event or are they just browsing? as users might spend a lot more than 2 minutes finding an event they wish to attend if they are simply browsing using the filters. Hope the feedback helps and good luck!

Matt

Figure 4: My comments to [redacted]

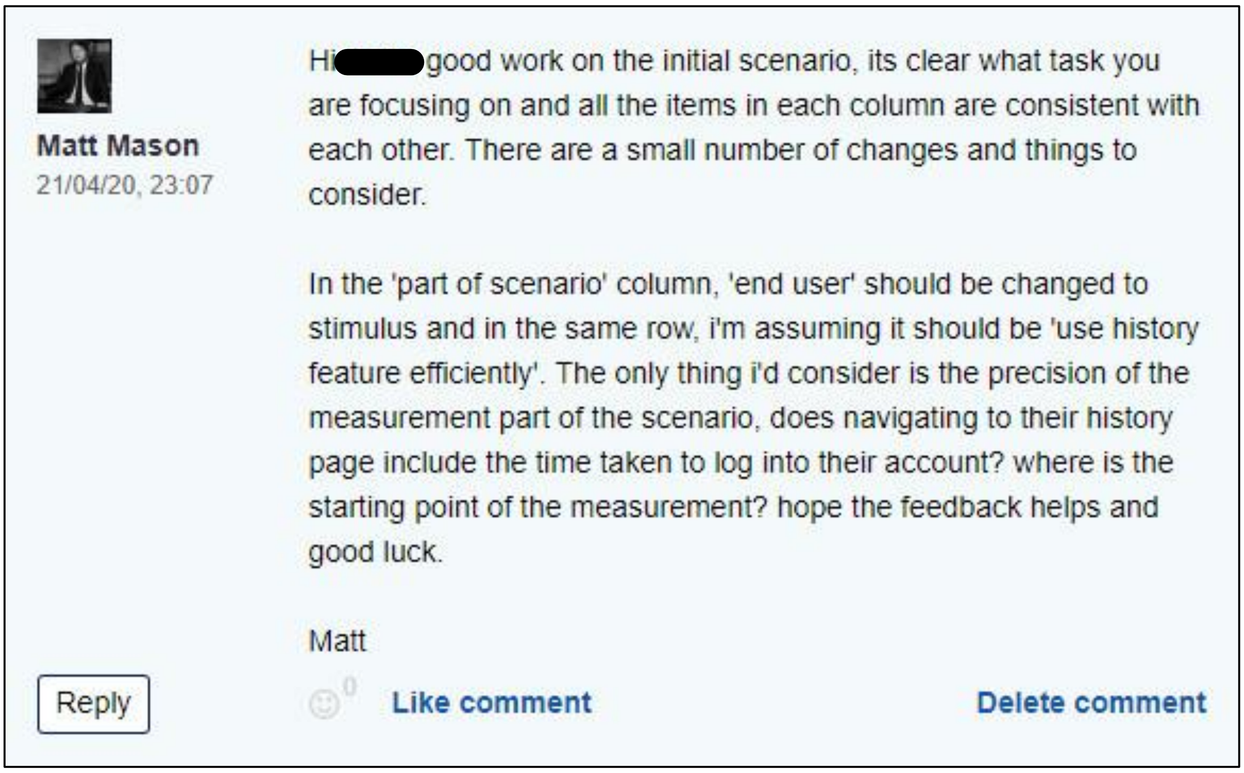


Usability scenario

Part of scenario	Type of Value	Actual Value
Source	End user	Member of the public
Stimulus	User wants to: - use system efficiently	Use BDSS system efficiently
Artefact	System	BDSS System
Environment	Using system	Normal system use
Response	Support efficient use	System allows individual to use filters to find an event
Measure	Task time	Time taken for an end user to find an event they wish to attend is 2 minutes

Figure 5: [redacted] usability scenario

My comments to [redacted]



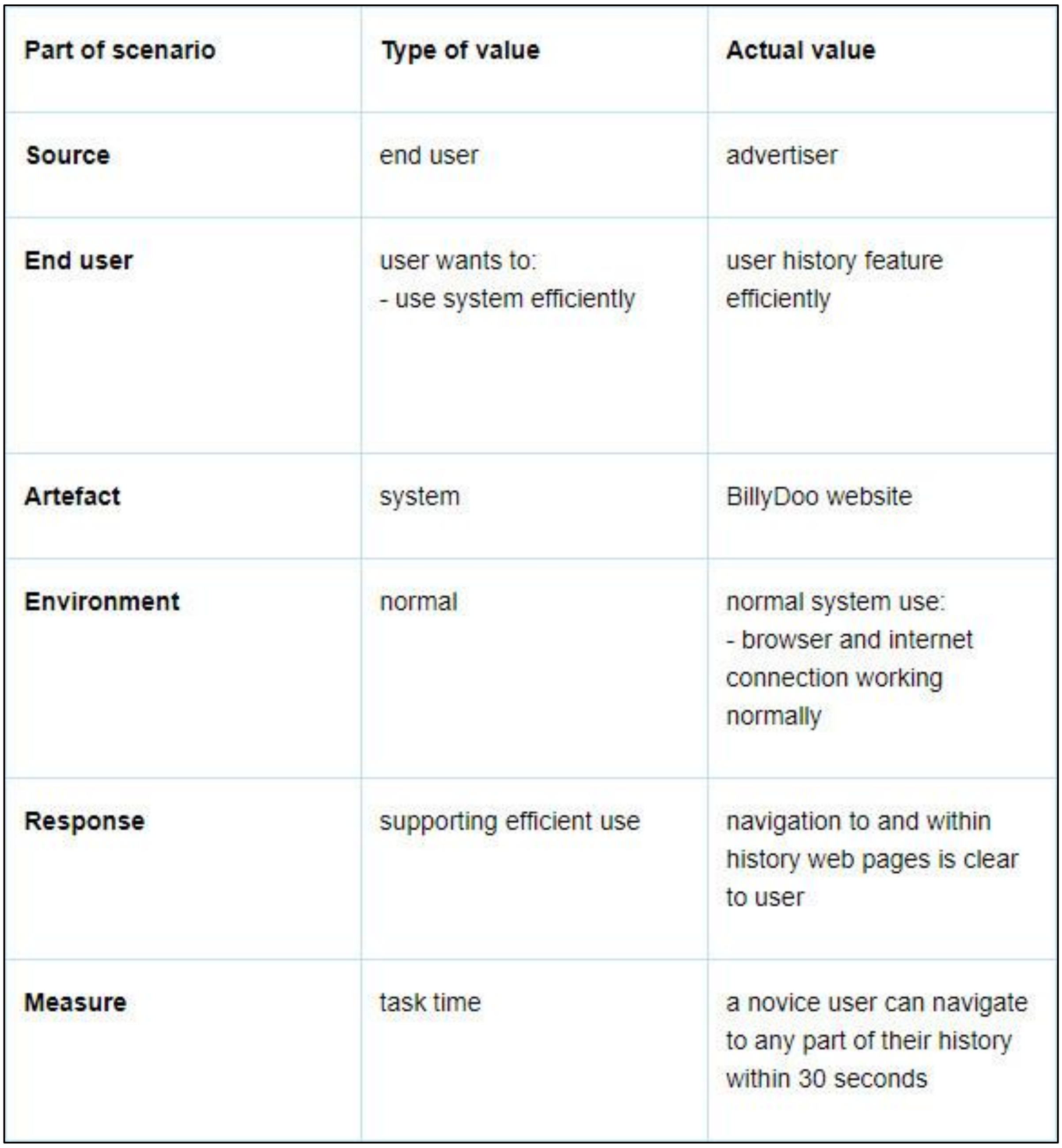
Matt Mason
21/04/20, 23:07

Hi [redacted] good work on the initial scenario, its clear what task you are focusing on and all the items in each column are consistent with each other. There are a small number of changes and things to consider.

In the 'part of scenario' column, 'end user' should be changed to stimulus and in the same row, i'm assuming it should be 'use history feature efficiently'. The only thing i'd consider is the precision of the measurement part of the scenario, does navigating to their history page include the time taken to log into their account? where is the starting point of the measurement? hope the feedback helps and good luck.

Matt

Figure 6: My comment to [redacted]



Part of scenario	Type of value	Actual value
Source	end user	advertiser
End user	user wants to: - use system efficiently	user history feature efficiently
Artefact	system	BillyDoo website
Environment	normal	normal system use: - browser and internet connection working normally
Response	supporting efficient use	navigation to and within history web pages is clear to user
Measure	task time	a novice user can navigate to any part of their history within 30 seconds

Figure7: [redacted] usability scenario

d.

Part of Scenario	**Type of Value**	**Actual Value**
Source	External	Web browser, internet user
Stimulus	Sporadic	Selecting username
Artefact	System	BillyDoo system
Environment	Normal	Browser and internet connection working normally
Response	Process stimuli	Return confirmation that username is acceptable or not
Response Measure	latency	Confirmation returned with 1 second of keyboard inactivity in 98% of cases provided the environment is working normally

My comments on improving my scenario: For my updated scenario I have chosen to keep things almost the same regarding the actual value for the part and type of scenario. Based on feedback from [redacted], I have made a slight change in the response measure which is to add some clarification to the value. Even under normal working conditions, data may be lost, or some issues may arise and cause the response to be slower than normal, so it helps to quantify the latency to provide a more defined quality requirement.

Question 4

a.

i.

The first step would be to establish the input data space for the method getCategory which returns the type of letter based on the following four inputs:

- Length (cm) – a number in the range 0.1 to 35.3
- Width (cm) a number in the range 0.1 to 25
- Thickness (cm) – a number in the range 0.1 – 2.5
- Weight (g) – a number in the range 0.1 - 750

The input data space for the method thus consists of all the possible combinations for four values taken from:

- {0.1, 0.2, …, 35.3}
- {0.1, 0.2, …, 25}
- {0.1, 0.2, …, 2.5}
- {0.1, 0.2, …, 750}

The next step would be to partition the data space into subdomains using use case analysis. As it is assumed design by contract is enforced, we need not worry about testing the precondition as the responsibility of not passing out-of-range data falls with the caller. When the dimensions of the letter are passed to the method, the type of letter is returned, either a letter, a large letter, or a parcel. This naturally leads to a partition of the input data space into three subdomains.

Letter: 0.1 ≤ Length ≤ 24 0.1 ≤ Width ≤ 16.5 0.1 ≤ Thickness ≤ 0.5 0.1 ≤ Weight ≤ 100

Large Letter: 24 < Length ≤ 35.3 16.5 < Width ≤ 25 0.5 < Thickness ≤ 2.5 100 < Weight ≤ 750

Parcel: 35.5 < Length 25 < Width 2.5 < Thickness 750 < Weight

The final step involves black-box testing subdomains given by the case analysis. For each subdomain, the test data will include expected data in the middle of each subdomain. It will also include boundary testing using values at the extreme ends of the subdomain and values close to the extreme. For example, we can establish 5 test cases for testing the size of a letter:

	Length	Width	Thickness	Weight
Letter (Extreme)	0.1	0.1	0.1	0.1
Letter (Close to extreme)	0.2	0.2	0.2	0.2
Letter (Typical)	14	8	0.3	50
Letter (Close to extreme)	23.9	16.4	0.4	99.9
Letter (Extreme)	24	16.5	0.5	100

However, there is also the situation where one argument value may be outside the range of what is expected causing the value returned to be not was expected. For example:

	Length	Width	Thickness	Weight

Letter (Length out of range)	25	8	0.3	50
Letter (Width out of range)	14	17	0.3	50
Letter (Thickness out of range)	14	8	1	50
Letter (Weight out of range)	14	8	0.3	150

There would also be four more test cases for the situation where one argument would be outside the range of both 'letter' and 'large letter' and return the string 'parcel'. Following this method, there should be 23 test cases overall.

- 5 for testing the size of a letter.
- 5 for testing the size of a large letter.
- 8 for testing arguments outside the expected range increasing the size by 1 subdomain (4 for the increase from letter to large letter and 4 for the increase of large letter to parcel)
- 4 for testing arguments outside the expected range increasing the size by 2 subdomains (from letter to parcel).
- 1 for testing the size of a parcel.

ii.

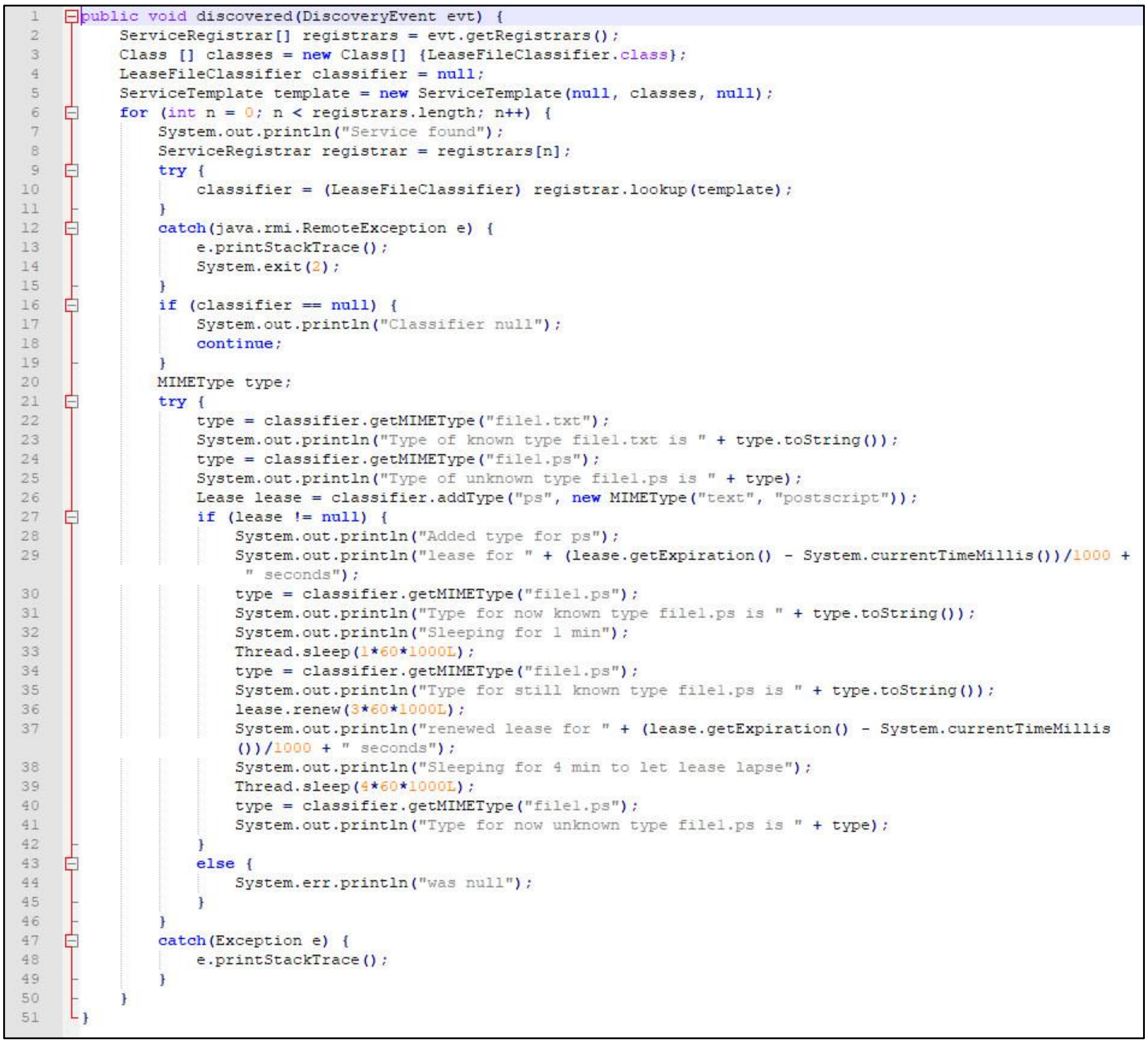
assert length > 0 && width > 0 && thickness > 0 && weight > 0;

b.

Unit testing involves a systematic testing of the methods used in a class combined with Test Driven Development, which suggests tests should be written before code implementation. Design by Contract simplify this by helping to design test cases and assertions by using the contracts pre and post conditions and writing test around it based on an acceptable set of inputs. This allows tests to be defined at an early stage, it helps define responsibility of checking functions, reduces code duplication, detects code errors, provides documentation, contributes to design, refactoring and the implementation is considered complete when all the tests pass.

Question 5

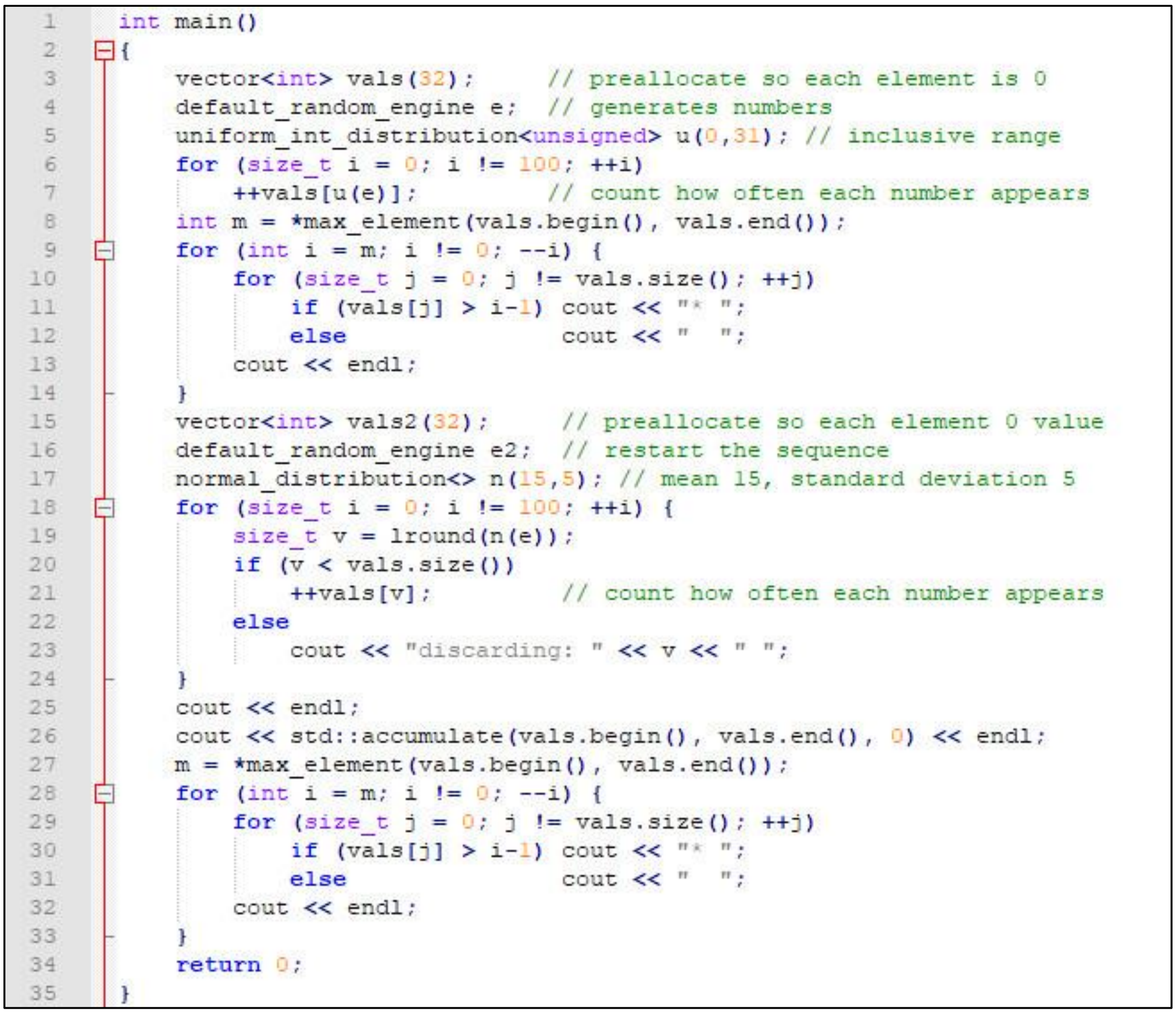
a.



```
public void discovered(DiscoveryEvent evt) {
    ServiceRegistrar[] registrars = evt.getRegistrars();
    Class [] classes = new Class[] {LeaseFileClassifier.class};
    LeaseFileClassifier classifier = null;
    ServiceTemplate template = new ServiceTemplate(null, classes, null);
    for (int n = 0; n < registrars.length; n++) {
        System.out.println("Service found");
        ServiceRegistrar registrar = registrars[n];
        try {
            classifier = (LeaseFileClassifier) registrar.lookup(template);
        }
        catch(java.rmi.RemoteException e) {
            e.printStackTrace();
            System.exit(2);
        }
        if (classifier == null) {
            System.out.println("Classifier null");
            continue;
        }
        MIMEType type;
        try {
            type = classifier.getMIMEType("file1.txt");
            System.out.println("Type of known type file1.txt is " + type.toString());
            type = classifier.getMIMEType("file1.ps");
            System.out.println("Type of unknown type file1.ps is " + type);
            Lease lease = classifier.addType("ps", new MIMEType("text", "postscript"));
            if (lease != null) {
                System.out.println("Added type for ps");
                System.out.println("lease for " + (lease.getExpiration() - System.currentTimeMillis())/1000 + " seconds");
                type = classifier.getMIMEType("file1.ps");
                System.out.println("Type for now known type file1.ps is " + type.toString());
                System.out.println("Sleeping for 1 min");
                Thread.sleep(1*60*1000L);
                type = classifier.getMIMEType("file1.ps");
                System.out.println("Type for still known type file1.ps is " + type.toString());
                lease.renew(3*60*1000L);
                System.out.println("renewed lease for " + (lease.getExpiration() - System.currentTimeMillis())/1000 + " seconds");
                System.out.println("Sleeping for 4 min to let lease lapse");
                Thread.sleep(4*60*1000L);
                type = classifier.getMIMEType("file1.ps");
                System.out.println("Type for now unknown type file1.ps is " + type);
            }
            else {
                System.err.println("was null");
            }
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 8: Code Sample A[1]

[1] (Newmarch, 1999 - 2005)



```cpp
int main()
{
    vector<int> vals(32);     // preallocate so each element is 0
    default_random_engine e;  // generates numbers
    uniform_int_distribution<unsigned> u(0,31); // inclusive range
    for (size_t i = 0; i != 100; ++i)
        ++vals[u(e)];         // count how often each number appears
    int m = *max_element(vals.begin(), vals.end());
    for (int i = m; i != 0; --i) {
        for (size_t j = 0; j != vals.size(); ++j)
            if (vals[j] > i-1) cout << "* ";
            else               cout << "  ";
        cout << endl;
    }
    vector<int> vals2(32);     // preallocate so each element 0 value
    default_random_engine e2;  // restart the sequence
    normal_distribution<> n(15,5); // mean 15, standard deviation 5
    for (size_t i = 0; i != 100; ++i) {
        size_t v = lround(n(e));
        if (v < vals.size())
            ++vals[v];         // count how often each number appears
        else
            cout << "discarding: " << v << " ";
    }
    cout << endl;
    cout << std::accumulate(vals.begin(), vals.end(), 0) << endl;
    m = *max_element(vals.begin(), vals.end());
    for (int i = m; i != 0; --i) {
        for (size_t j = 0; j != vals.size(); ++j)
            if (vals[j] > i-1) cout << "* ";
            else               cout << "  ";
        cout << endl;
    }
    return 0;
}
```

Figure 9: Code Sample B[2]

b.

Sample A: LOC Metric = 51
Sample B: LOC Metric = 35

Sample A: Cyclomatic-Complexity Metric = 6
Sample B: Cyclomatic-Complexity Metric = 10

Line Number	Cyclomatic Complexity
1	1 (starts at 1)
6	2 (for statement)
9	3 (try statement – single catch line 12)
16	4 (if statement)
21	5 (try statement – single catch line 47)
27	6 (if statement)

[2] (Stanley B. Lippman, 2014)

Line Number	Cyclomatic Complexity
1	1 (starts at 1)
6	2 (for statement)
9	3 (for statement)
10	4 (for statement)
11	5 (if statement)
18	6 (for statement)
20	7 (if statement)
28	8 (for statement)
29	9 (for statement)
30	10 (if statement)

c.

Sample code A consists of 51 lines of code, whereas sample code B consist of 35 lines of code. The complexity according to the LOC metric is that sample code A should be more complex, however if we use the cyclomatic complexity metric, its suggests that sample code A has a complexity of 6 and sample code B has a complexity of 10 suggesting sample code B is more complex of the two. This supports that sample code B is more complex than sample code A.

d.

For each of the examples given above, sample code B may benefit from some restructuring of code into separate units. As a rule of thumb, any code that has a complexity of 10 or greater may benefit from altering the code in some way such as writing helper methods. As sample code B has a cyclomatic complexity of 10 it may benefit from restructuring.

References

Newmarch, J., 1999 - 2005. *Chapter 15: More complex Examples.* [Online]
Available at: https://jan.newmarch.name/java/jini/tutorial/MoreComplex.html
[Accessed 25 04 2020].

Stanley B. Lippman, J. L. B. E. M., 2014. *C++ Primer Fifth Edition.* 5th ed. s.l.:Addison-Wesley.