User Stories Of Car Insurance X problem

1. **Authentication**: User should be authenticated

(Hours: 1)

1. **UserGroups**: Logged in users are categorized in Car Department A (Higher Ranked Employee), Car Department B, Finance Department

(Hours: 2)

1. **CustomerForm**: Customer fills in accident form with Name, Email, Phone, License Plate and accident description

(Hours: 1.5)

1. **FormPreview**: Employees at CD are able to see submitted accident forms

(Minutes: 30)

1. **FormRegistration**: If customer is client employees are able to register a submitted claim

(Minutes: 30)

1. **ClientCheck**: System checks if customer is a client

(Minutes: 30)

1. **ClientHistory**: System retrieves client history, if any

(Minutes: 30)

1. **ClaimRejection**: If customer is not client, CDA rejects the claim

(Minutes 20)

1. **CategorizeComplex**: If the client has a history, or the car value is over $15000, the system categorizes the claim as complex

(Minutes: 30)

1. **CategorizeSimple:** If the client doesn’t have a history and the car value is less than $15000, the system categorizes the claim as simple.

(Minutes: 30)

1. **GarageQuery**: System sends an email to the garage with the licence plate of the car of the client to get the cost of the repair

(Hours: 1)

1. **GarageForm**:The garage fills a form with the cost and the licence plate.

(Hours: 1)

1. **ClaimAcceptance**: CDA decides if the claim should be paid or not.

(Minutes: 30)

1. **ClaimArchive**: If CDA decides that the claim should not be paid, the claim is archived

(Minutes: 30)

1. **ClaimPay**: If CDA decides that the claim should be paid, he marks “to be paid”

(Minutes: 30)

1. **ViewClaimsToPay**: Finance department employees can see claims marked “to be paid”

(Minutes: 30)

1. **ClaimPaid**: Finance department employees pay the claim, after that the claim is archived.

(Minutes: 30)

1. **ClientSearch**: Any logged in user can search using the name, email, phone or licenceplate of the client to see if there is a claim in progress and what stage it is.

(Hours: 2)

1. **PaidHistory**: Finance department employees can see the archive of paid claims.

(Hours: 1)

1. **ClientInformSMS**: Client should be informed for every status change via automated SMS

(Hours: 3)

1. **PrintPDF**: Forms should be printable in PDF

(Hours: 1)

1. **ExportExcel**: Economic records should be exported in spreadsheets

(Minutes: 30)

1. **ClaimSort**: Claims should be presented sorted by registration date or Client name

(Hours: 1)

1. **LogoutInactive**: System should log out inactive users

(Minutes: 20)

1. **PersonalRecordsLog**: System should keep record of personal records of customer

(Hours: 1)

1. **InsuranceRecordsLog**:System should keep record of car insurance records

(Hours: 1)

1. **ClientDocumentsLog**: System should keep record of related documents

(Hours: 1)

1. **ClientClaimHistory**: System should keep record of history of previous claims

(Hours: 1)

1. **CrossPlatform**: System should be able to be executed in Windows, \*nix

(Hours: 1)

1. **AddUsers**: CDA should be able to add new users with roles to the system

(Hours: 2)

1. **ChangeUserGroups**: CDA should be able to change user roles to the system

(Hours: 2)

1. **EmployeeProfile**: Employees will be able to review their personal data

(Hours: 1)

1. **AddClient**: Finance department will be able to register new clients in the system

(Hours: 1)

1. **AuthenticationLog**: Authentications should be logged in a logging system

(Hours: 1.5)

1. **ActionLog**: All actions should be logged in a logging system

(Hours: 1).

1. **Help**: At every stage help should be available for the employees

 (Hours: 3)

1. **Contracts**: Finance Department employees can create car insurance contracts for a client.

(Hours: 1).

|  |  |  |  |
| --- | --- | --- | --- |
|  | High Value | Medium Value | Low Value |
| High Risk | ClaimAcceptance (2)  ViewClaimsToPay (2)  AddClient (3) | ClaimArchive (2) | ClaimPaid (1),  *ClaimSort* (1)  *PersonalRecordsLog* (1)  *InsuranceRecordsLog* (1)  *ClientInformSMS* (2)  *ClientDocumentsLog* (1)  *AddUsers* (2)  *ChangeUserGroups* (2)  *ActionLog* (1)  *Help* (3) |
| Medium Risk | UserGroups (3)  FormRegistration (1)  ClaimPay (1)  ClientClaimHistory (2) | FormPreview(1)  CategorizeComplex(1)  CategorizeSimple (1)  *Contracts(1)* | GarageQuery (1)  GarageForm (1)  ClientSearch (2)  *PrintPDF* (1)  *ExportExcel* (1)  *EmployeeProfile* (1)  *AuthenticationLog* (2) |
| Low Risk | Authentication (1)  ClientCheck (1)  ClientHistory (1) | CustomerForm (2)  ClaimRejection (1) | PaidHistory (1)  *LogoutInactive* (1)  CrossPlatform (1) |

According to the Table above, the User Stories we selected to implement in the first release are those who are underlined. The reason we selected all the High Value user stories is that they represent the main functionality of the system. User stories with Low Value, especially those with High Risk, will be implemented in future releases, since they are not that valuable to the client and will need a lot of time to implement. It is visible from the Table that we have implemented user stories that reside in the upper left corner (High Value and High Risk).

|  |  |
| --- | --- |
| **Iteration Plan for First Release** | |
| *Story Points* | *Story Name* |
| Iteration 1 | |
| 1 | Authentication |
| 3 | UserGroups |
| 2 | CustomerForm |
| 1 | FormPreview |
| 1 | FormRegisration |
| Iteration 2 | |
| 3 | AddClient |
| 1 | ClientCheck |
| 1 | ClientHistory |
| 2 | ClientClaimHistory |
| 1 | CategorizeSimple |
| 1 | CategorizeComplex |
| Iteration 3 | |
| 1 | GarageQuery |
| 1 | GarageForm |
| 2 | ClaimAcceptance |
| 1 | ClaimRejection |
| 2 | ClaimArchive |
| 1 | ClaimPay |
| Iteration 4 | |
| 2 | ViewClaimsToPay |
| 1 | ClaimPaid |
| 2 | ClientSearch |
| 1 | PaidHistory |
| 1 | CrossPlatform |
| 1 | PersonalRecordsLog |

**Metaphor**

**Test-Driven Programming**

At first, the idea that we had to implement tests for each functionality in the system before the functionality itself was new to us and we couldn’t see how it could benefit us. When we started writing the first lines of codes with that technique, we saw that it made our lifes easier and in many cases we detected errors that otherwise would be hard to find. It is true that in very simple functions it felt dumb to write tests, but we tried to stick to that logic and we applied the test-driven programming process until the end.

**Pair Programming**

Pair programming was also a new programming approach for both of us. After three days of meeting we both agree that two minds are better than one. A lot of times one assisted the orther even for minor faults like a typo mistake or even better provided a new radical idea on how to implement a feature. Roles changed quite frequiently during the day allowing one to get some physical rest and try to approach the problem from a different angle. A few times when we implemented the GUI we diverted from the aspects of pair programming in terms that we were both coding at the same time but still exchanging thoughts and asking each other for advice if needed. Overall it was a very good experience but the ongoing courses and short deadlines didn’t let us enjoy it as much as we could.

**Refactoring**

Refactoring of our code was part of each iteration. After testing, implementing and re-testing a feature of our system we looked more thoroughly in our code. Very often when you are writing code you do not pick the best approach to a problem. You end up writing the same code twice or using too many objects which will eventually have an impact on readability and performance. Thus we adopted refactoring with pleasure although we did not write a “real” system. Most of our refactoring dealt with object reusability, assigning null value to objects in the right position in order to give a hint to the Garbage Collector, refactoring pieces of code that were used frequently into methods and splitting ones with too many loc.

**Estimation Assessment**

**Acceptance Tests**

**Daily stand-up meetings**