Car Insurance

Project for Modern Methods of Software Engineering Course – ID2207

Antonios Kouzoupis - Nikolaos Ektoras Anestos

October 16, 2014
antkou@kth.se
anestos@kth.se

Introduction

The Insurance Company X system will be able to handle claims for car accidents. There are three kinds of possible users of the application. Simple users that can report an accident using a form, employees who have three different roles and garages the company collaborates with. Simple users can fill up a form with details of the accident. Low ranked Car Damages employees see those forms and register them as payment claims (if the claimant is a client of the company) and ask the garage about the cost of the repair. Higher ranked Car Damages employees can see the registered claims and choose to pay them or not. Finance department employees see the claims they should pay and act accordingly. All the claims are stored in an archive and all the company employees can search for the history of a client.

User Stories Of Car Insurance X project

1. **Authentication**: User should be authenticated
   (Hours: 1)
2. **UserGroups**: Logged in users are categorized in Car Damage Department A (Higher Ranked Employee), Car Damage Department B, Finance Department
   (Hours: 2)
3. **CustomerForm**: Customer fills in accident form with Name, Email, Phone, Licence Plate and accident description
   (Hours: 1 1/2)
4. **FormPreview**: Employees at CD are able to see submitted accident forms
   (Minutes: 30)
5. **FormRegistration**: If customer is client employees are able to register a submitted claim
   (Minutes: 30)
6. **ClientCheck**: System checks if customer is a client
   (Minutes: 30)
7. **ClientHistory**: System retrieves client history, if any
   (Minutes: 30)
8. **ClaimRejection**: If customer is not client, CDA rejects the claim
   (Minutes 20)

9. **CategorizeComplex**: If the client has a claim history, or the car value is over $15000, the system categorizes the claim as complex
   (Minutes: 30)
10. **CategorizeSimple:** If the client doesn't have a claim history and the car value is less than $15000, the system categorizes the claim as simple.
    (Minutes: 30)
11. **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 when the claim is registered
    (Hours: 1)
12. **GarageForm**: The garage fills a form with the cost and the licence plate.
    (Hours: 1)
13. **ClaimAcceptance**: CDA decides if the claim should be paid or not.
    (Minutes: 30)
14. **ClaimArchive**: If CDA decides that the claim should not be paid, the claim is archived
    (Minutes: 30)
15. **ClaimPay**: If CDA decides that the claim should be paid, he marks "to be paid"
    (Minutes: 30)
16. **ViewClaimsToPay**: Finance department employees can see claims marked "to be paid"
    (Minutes: 30)
17. **ClaimPaid**: Finance department employees pay the claim, after that the claim is archived.
    (Minutes: 30)
18. **ClientSearch**: Any logged in user can search using the name, email, phone or licence plate of the client to see if there is a claim in progress and what stage it is.
    (Hours: 2)
19. **PaidHistory**: Finance department employees can see the archive of paid claims.
    (Hours: 1)
20. **ClientInformSMS**: Client should be informed for every status change via automated SMS
    (Hours: 3)
21. **PrintPDF**: Forms should be printable in PDF
    (Hours: 1)
22. **ExportExcel**: Economic records should be exported in spreadsheets
    (Minutes: 30)
23. **ClaimSort**: Claims should be presented sorted by registration date or Client name
    (Hours: 1)
24. **LogoutInactive**: System should log out inactive users
    (Minutes: 20)
25. **PersonalRecordsLog**: System should keep record of personal records of customer
    (Hours: 1)
26. **InsuranceRecordsLog**: System should keep record of car insurance records
    (Hours: 1)
27. **ClientDocumentsLog**: System should keep record of related documents
    (Hours: 1)
28. **ClientClaimHistory**: System should keep record of history of previous claims

(Hours: 1)

29. **CrossPlatform**: System should be able to be executed in Windows, *nix
    (Hours: 1)
30. **AddUsers**: CDA should be able to add new users with roles to the system
    (Hours: 2)
31. **ChangeUserGroups**: CDA should be able to change user roles to the system
    (Hours: 2)
32. **EmployeeProfile**: Employees will be able to review their personal data
    (Hours: 1)
33. **AddClient**: Finance department will be able to register new clients in the system
    (Hours: 1)
34. **AuthenticationLog**: Authentications should be logged in a logging system
    (Hours: 1 1/2)
35. **ActionLog**: All actions should be logged in a logging system
    (Hours: 1)
36. **Help**: At every stage help should be available for the employees
    (Hours: 3)
37. **Contracts**: Finance Department employees can create car insurance contracts for an existing client.
    (Hours: 1)
38. **CheckRegisteredClaims**: The system should not allow CD employees to register an accident form as a claim when there is already a claim unhandled for the same client.
    (Minutes: 30)

Risk – Value assessment

	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)* CheckRegisteredClaims(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 "cost" more 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	CheckRegisteredClaims
1	ClaimPay
Iteration 4	
2	ViewClaimsToPay
1	ClaimPaid
2	ClientSearch
1	PaidHistory
1	CrossPlatform
1	PersonalRecordsLog

Metaphor

Railway: A defined set of stations (actions) must be reached in order to reach our destination (handle a claim) using predefined paths. The train (claim) leaves for the next station when the driver (user) selects to.

Acceptance Tests

Acceptance test for **CustomerForm** user story
(Customer fills in accident form with Name, Email, Phone, Licence Plate and accident description)

A user who wants to submit a car accident form, presses the button "Report an Accident". The form is visible to him and he can fill up the fields with appropriate data. In order to submit the form, he should press the submit button. When the form is submitted, the user is informed that the data are now stored in the application.

Acceptance test for **ViewClaimsToPay** and **ClaimPaid** user stories
(Finance department employees can see claims marked "to be paid", Finance department employees pay the claim, after that the claim is archived.)

A finance department employee logs in the application with his user account and password. A list of claims that he has to pay is presented to him. He can select claims from that list and press the "Pay" button to pay that claim using online banking. The paid claim is removed from the list. At any point, he can press "View Payment History" button, which presents a list of claims that the company has paid.

Daily stand-up meetings

Since we broke up the problem to 4 iterations, we tried to implement each iteration in a day of coding. We agreed that each day we finish, we would commit the code to git so we could easily check what was done in each iteration.

In our first meeting, we discussed the problem and broke it up to several user stories (derived from the text) and added some user stories that we believed would be reasonable for

this type of application. We also did some research on how to apply extreme test-driven programming since we didn't have any previous experience with that technique.

In the second meeting, we talked about what we prepared the first day and decided that we will implement the user stories from the first iteration that day.

The third meeting was an actual "stand-up" meeting because we could talk over what was accomplished the previous day and discuss of any problems that we might have. The previous day we implemented Authentication, UserGroups, CustomerForm, FormPreview, FormRegisration user stories from the first iteration. Not everything went as planned, because the FormPreview user story took us more time than estimated because we didn't have that much experience in building gui. We also agreed that we would continue with the second iteration.

The next meeting we talked about the second iteration, how we could avoid the problems that delayed us the previous days (such as exporting a list of objects to JTable) and prepared about the user stories of the third iteration.

The fifth meeting we discussed about refactoring, since the last day we used quite some time for it, and how we could "refactor on the fly" in order to be more productive. We also discussed about the order we will implement the user stories of the last iteration. Finally, we discussed what the acceptance tests would be.

Estimation Assessment

The first estimation of the four iterations we chose to implement was roughly 20hours. We managed to finish the project more or less within that timeframe, but in some cases the estimation was wrong. For example in FormPreview user story, we used more time than we first estimated in contrast on cases like the user story AddUsers where we used less. The total time we allocated for the assignment is almost 30 hours including the time we used to write this report, break the problem to user stories and plan the iterations.

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 lives 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.

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 lines of code.

Pair Programming

Pair programming was also a new programming approach for both of us. After five days of meetings we both agree that two minds are better than one. A lot of times one assisted the other even for minor faults like a typo mistakes or even better provided a new radical idea on how to implement a feature. Roles changed quite frequently 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.