Risk Analysis Tool Design and Implementation Document

Yan CHEN, ychence@connect.ust.hk
Xiang GAO, xgaoaa@connect.ust.hk
Tian XIE, txieaa@connect.ust.hk
Fang XIONG, fxiongaa@connect.ust.hk

Version	Date	Changes
1.0	February 15, 2015	Initial version (word format)
2.0	April 19, 2015	Migrate from word version
2.1	May 3, 2015	Analytic UCVA engine design; FVA computation discussion; default models discussion
3.0	May 17, 2015	Test cases discussion in appendix

ACKNOWLEDGMENTS

The authors would like to thank Dr. Chak WONG, the instructor of the MAFS 5220 course, and Felix LEE, the teaching assistant, for providing guidance of the project.

1. INTRODUCTION

In OTC market, abundant financial products are traded frequently and the price is always concerned by each trader. In traditional, risk neutral approach is applied to get arbitrage price of financial contracts. One of the problems is that no-arbitrage price is just a default free price which assumes all entities will fulfill its obligation. However, counterparty default risk widely exists in OTC market so taking counterparty risk into consideration is necessary when one tries to price an OTC product C this leads to the computation of CVA family, including CVA, DVA, FVA, etc.

Besides from the pricing point of view to analysis counterparty risk, financial institution want to integrate all results into a simple number to indicate how much reserves they should put aside which this leads to computations of VaR and Credit VaR. These requirements motivates us to build an application to compute the counterparty risk factors like Credit VaR, CVA, DVA and FVA, with Wrong Way Risk (WWR) and collateral into consideration.

Risk Analysis Tool is a platform that we first attempt to make it into use. We create the computation supporting Interest Rate Swap, Equity Swap and Cross Currency Swap products, and a portfolio with them. The computing method is determined case by case, either using analytical solution, or Monte Carlo method. Given the fact that intensive computation such as Monte Carlo is frequently needed, we create the platform as a cloud service that would potentially be accessed via various ways, including web, mobile application, and desktop clients. Our initial version includes a mobile application for Android platform. Support for other client-side access would be done in future. Our pricing components utilize the QuantLib and our own implementation tries to be consistent with the QuantLib design. We plan to make our extensions be a part of the next version of QuantLib.

The rest of the document is organized as follow: Section 2 introduces necessary background knowledge for counterparty credit risk and funding valuation adjustment, Section 3 defines the scope of the project including the products and features to be covered, Section 4 represents the design of the compute engine, Section 5 discusses important implementation details, traps and pitfalls in the system, Section 2 discusses about the testing methodology that ensures the quality of our project.

For more details of our implementation including the API and class reference and testing status, additional documents are provided as appendixes.

2. BACKGROUND

For better understanding of the document and our project for those who has no background knowledge about counterparty risk and CVA family adjustment, we introduce the necessary background knowledge for better understanding the remaining parts of the document, including the basis of counterparty credit risk and various adjustments like CVA, DVA and FVA.

2.1. Counterparty Risk for Derivative Transactions

The counterparty risk means the risk to each party of a contract that the counterparty will not live up to its contractual obligations. Counterparty risk as a risk to both parties and should be considered when evaluating a contract.

Assessing the credit risk for a derivatives transaction is much more complicated than assessing the credit risk for a loan because the future exposure (i.e., the amount that could be lost in the event of a default) is not known. Derivatives that trade on exchanges entail very little credit risk because exchange stands between the two parties and has strict rules on the margin to be posted by each side.

2.2. Definition of Exposures

Definition 2.1 (*Exposure at time* t). For a position with final maturity T and whose discounted and added random cash flows at time $t < T$ are denoted by $\Pi(t, T)$, the exposure at time t is defined as

$$Ex(t) = \mathbb{E}_t^Q[\Pi(t, T)^+]$$

Definition 2.2 (*Exposure at time* t *with sign*). For a position with final maturity T and whose discounted and added random cash flows at time $t < T$ are denoted by $\Pi(t, T)$, the exposure at time t with sign is defined as

$$Exs(t) = \mathbb{E}_t^Q[\Pi(t, T)]$$

Definition 2.3 (*Expected exposure at time* t). For a position with final maturity T and whose discounted and added random cash flows at time $t < T$ are denoted by $\Pi(t, T)$, the expected exposure at time t is defined as

$$EEx(t) = \mathbb{E}_t^P[Ex(t)] = \mathbb{E}_t^P[\mathbb{E}_t^Q[\Pi(t, T)]^+]$$

Please note that the outer expectation is taken under the physical measure.

Definition 2.4 (*Exposure at default*). The expected exposure at default is simply the exposure at default time τ

$$EAD = Ex(\tau) = \mathbb{E}_t^Q[\Pi(t, T)]^+$$

Please note that the outer expectation is taken under the physical measure.

2.3. CVA, DVA and FVA

CVA family charges, including Credit Value Adjustment (CVA), Debt Value Adjustment (DVA), and Bilateral Value Adjustment which is the difference of CVA and DVA. If we consider funding cost, we will adjust the BVA with Funding Value Adjustment, which is the Total Value Adjustment (TVA). The adjustment is then added into the default-free price of the contract to reflect the counterparty credit risk charge.

CVA is defined as the difference between the value of a position traded with a default-free counterparty and the value of the same position when traded with a given counterparty. If we assume the investor to be default-free, the CVA is called unilateral CVA (UCVA).

Definition 2.5 (*Unilateral CVA*).

$$UCVA = \mathbb{E}_0^Q[(1 - REC)D(0,\tau)\mathbf{1}_{\{\tau<T\}}Ex(\tau)]$$

If we assume both parties can default, we can default bilateral Counterparty Credit Risk, Debt Value Adjustment (DVA), and Bilateral Value Adjustment (BVA)

Definition 2.6 (*Bilateral CVA and DVA*).

$$CVA = \mathbb{E}_0^Q[(1 - REC)D(0,\tau)\mathbf{1}_{\{\tau_{1st}=\tau_C<T\}}Ex(\tau)^+]$$

$$DVA = \mathbb{E}_0^Q[(1 - REC)D(0,\tau)\mathbf{1}_{\{\tau_{1st}=\tau_B<T\}}Ex(\tau)^-]$$

According to [Damiano Brigo and Pallavicini 2013], the FVA is because funding the hedging of the derivative cannot be done by using a single risk-free rate. We cannot compute it independently and then add it to the default-free price due to the recursive nature of FVA - the hedging depends on the valuation and the valuation is derived from the hedging.

Then we could adjust the price of the contract relative to the default-free price

$$\bar{V} = V - CVA + DVA + FVA$$

3. PROJECT SCOPE

In this project, we aim to compute the counterparty valuation adjustment (CFA, DVA) and funding valuation adjustment (FVA) for a portfolio with or without correlations. Due to the complexity of the derivative products, we make strict assumptions in order to limit the scope and efforts of our project.

3.1. Major Components

In general, our system is designed as two parts - a compute engine and a mobile application. The computing engine contains instruments, pricing engines, stochastic processes. The functionalities are exposed via RESTful web service. The mobile application is used to collect the user input, invoking the web service, and display the results to the user when the computation completes.

Figure 1 represents the general architecture of the system, basically two parts - cloud service for compute engine and the mobile application

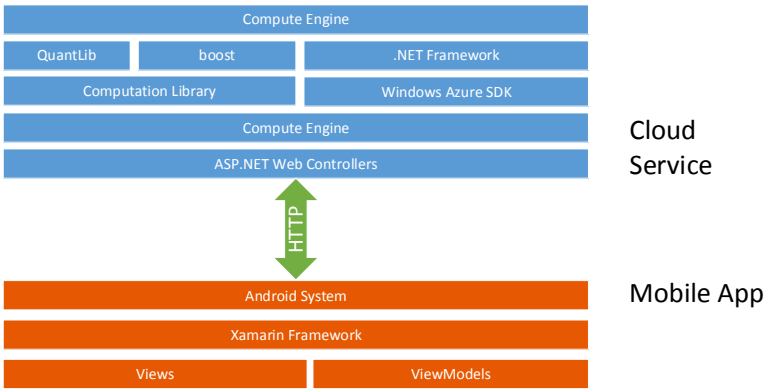


Fig. 1. System Architecture

3.2. Products

We focus on handling a portfolio containing one or more *Interest Rate Swap (IRS)*, *Equity Swap* and *Cross Currency Swap (CCS)* instruments.

Interest rate swap is a contract in which two parties agree to exchange interest rate cash flows, based on a specified notional amount from a fixed rate to a floating rate (or vice versa) or from one floating rate to another. In this part, we only consider the most common type of interest rate swap: plain vanilla swap. It often exchanges a fixed payment for a floating payment, which is linked to an interest rate (most often the LIBOR plus a spread), with predefined exchange schedules under the same currency. In particular, the schedules of the two parties could not be the same.

Equity swap is a contract to specify two parties exchange fixed rate coupon and equity dividends in the future. At the start, the payer side give a number of stocks to the receiver side, in exchange for the equivalent amount of dollar money from the receiver side; this amount is called the notional amount. Since the stocks have the value equal to the notional on the start date, there is in fact no change of value at the start. On every payment date before the maturity, the payer side pays the fixed rate coupon and receives the equity dividends cumulated in that period; while the receiver

side pays the dividends and receives fixed rate coupon. At maturity, in addition to the coupon and dividends exchange, the notional and the stocks are also swaped back.

Cross currency swap is a contract in which two parties exchange cumulative interest rate coupons dominated in different currencies. At the initiation, the frequency of payment and two fixed coupon rates are specified; also two parties exchange the notional dominated in different currency. Essentially, there is no upfront change of value at start. At intermediate payment dates, two parties exchange fixed rate coupons based on pre-determined interest rates. At maturity, the two parties exchange the nominal value back. Since the variability of exchange rate, the money paid by the two parties should have different worth at that time. In our project, cross currency swap is implemented as the fixed-fixed case which means the coupon rate paid by two legs are pre-defined at the beginning and will never change during the life of the contract.

As we compute the value adjustments are a portfolio including the products discussed above. There could be correlation among the price movements of these products. We also allow correlations between product prices and the counterparty defaults, so wrong way risk (WWR) and right way risk (RWR) can be dealt with.

3.3. Counterparty Credit Valuation Adjustments

In our projects, we cover the computation of various counterparty credit valuation adjustment, including the unilateral CVA, bilateral CVA and DVA. We use risk-free close-out assumption in our entire implementation.

We support computing unilateral CVA for a single IRS, Equity Swap or CCS instrument without Wrong Way Risk using analytic or synthetic solutions. For other cases, Monte Carlo method is adopt in computing all the valuation adjustments.

3.4. Funding Valuation Adjustment

According to [Damiano Brigo and Pallavicini 2013], the definition of FVA is not as straightforward as CVA or DVA, and we cannot compute it independently due to the recursive nature of FVA. We have to use American Monte Carlo if we adopt the method.

However, In our project, we plan to adopt a different way of computing FVA. According to [Schwietert 2014], assuming that the local currency cash to be the only valid collateral, we can compute the FVA as follows

$$FVA = -\mathbb{E}_0^Q[\int_0^T 1_{\{\tau_{1st}>t\}} D(0,t)\gamma(t)(V(t,T)^+ + V(t,T)^-)dt]$$

where $V(t,T)$ is the exposure at time t. And this enables us to compute FVA using regular Monte Carlo method.

4. COMPUTE ENGINE

The computing engine, which is most fundamental part, is based on the QuantLib[QuantLib 2015]. We define new **instrument classes** if not exist in QuantLib. The instruments include the Equity Swap and Cross Currency Swap mentioned in Section 3, also BVA and TVA which is treated as prices. We create new **pricing engines** for existing and new instruments, including ones using analytical solutions and Monte Carlo methods. Computation of exposure at default time for specific instruments is isolated into **instrument models** and **counterparty models**.

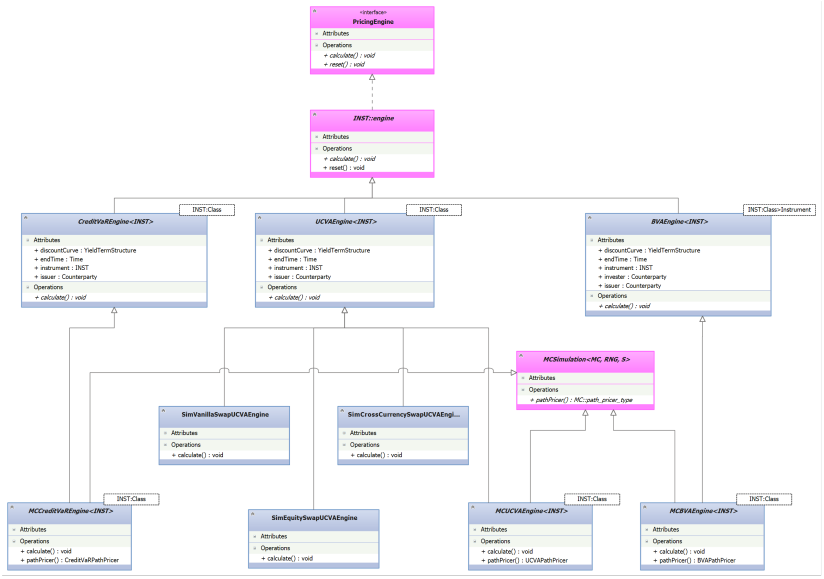
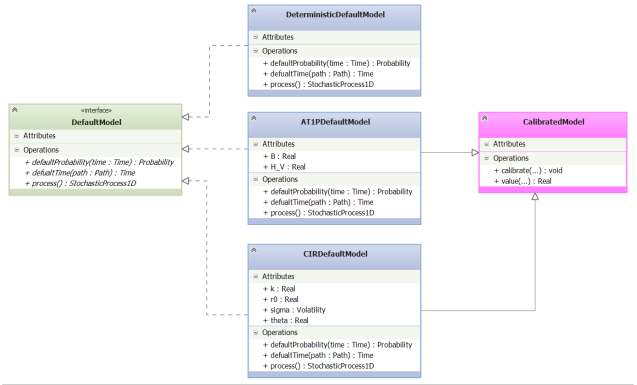
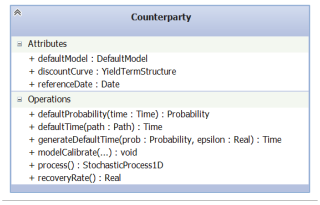


Fig. 2. Pricing Engines



(a) Default Models



(b) Counterparties

Fig. 3. Default and Counterparty Models

4.1. Default Models and Counterparties

Default models usually require calibration using market data. As a result, we implement the default model using the calibrated model framework in QuantLib. The Counterparty class, which will be used by the BVA and TVA engine, will hold a reference to a specific default model. In addition, the counterparty object have information of recovery rate and default time information related to the counterparty risk. The class diagrams of default models and counterparty are shown in Figure 3(a) and Figure 3(b).

The **Analytically-Tractable First Passage (AT1P) Model** [Damiano Brigo and Pallavicini 2013], which is a firm value (or structural) model, assumes the risk neutral dynamics for the value of the firm V is characterized by a risk-free rate r_t, a payout ratio k_t and an instantaneous volatility σ_t, according to the equation:

$$dV_t = V_t(r_t - k_t)dt + V_t\sigma_t dW_t$$

and assume a default barrier $H(t)$ of the form

$$H(t) = Hexp(\int_0^t (r_u - k_u - B\sigma_u^2)du)$$

where H and B are parameters.

The **Cox-Ingersoll-Ross (CIR) Model**, which is a intensity model that modelling default intensity using short-rates, assumes

$$dr_t = k(\theta - r_t)dt + \sigma\sqrt{r_t}dW_t$$

where

$$2k\theta > \sigma^2$$

4.2. General Monte Carlo Engines

For computation of BVA and TVA using Monte Carlo, the only part depend on the specific instrument is the exposure and underlying process, as a result, we externalize the exposure computation into underlying model, the counterparty default modelling to the counterparty and default model, and leave other parts to a shared Monte Carlo engine. For example, recall the computation of bilateral CVA, DVA and FVA

$$CVA = \mathbb{E}_0^Q[(1 - REC)D(0,\tau)\mathbf{1}_{\{\tau_{1st}=\tau_C<T\}}Ex(\tau)^+]$$

$$DVA = \mathbb{E}_0^Q[(1 - REC)D(0,\tau)\mathbf{1}_{\{\tau_{1st}=\tau_I<T\}}Ex(\tau)^-]$$

$$FVA = -\mathbb{E}_0^Q[\int_0^T 1_{\{\tau_{1st}>t\}}D(0,t)\gamma(t)(V(t,T)^+ + V(t,T)^-)dt]$$

The counterparty provides the recovery rate and determines which counterparty defaults (if any), and the Monte Carlo BVA engine handle all the computing workflow, such as discounting, checking first-to-default condition, and taking the expectation of all paths. The only instrument-specific part is the computation of exposure $Ex(\tau)$. The class diagram of Monte Carlo exposure models are shown in Figure 4

By externalizing the instrument-specific parts, we are also able to support computing the BVA and FVA for a portfolio rather than just a single position. Specifically, the general Monte Carlo engine simulates the underlying processes with correlation in consideration, then for each path, it invokes the instrument model separately to get the exposure for each single instrument. After getting all the single instrument exposures, it aggregate them according to the netting rules. Then the other parts of the computing the BVA and FVA are not affected.

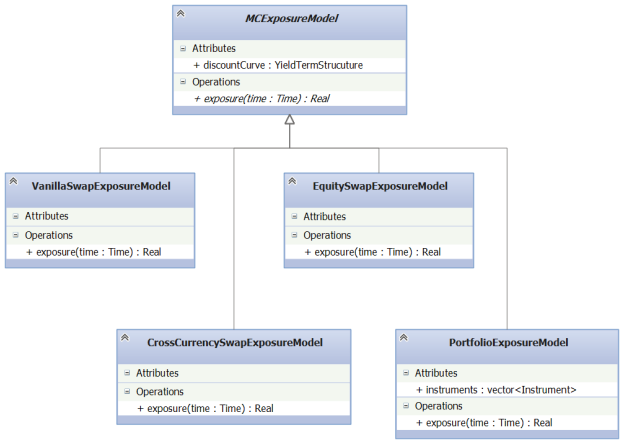


Fig. 4. MC Exposure Models

4.3. Instrument-Specific Unilateral CVA Engines

For some simple cases, it's possible that we can compute the CVA, DVA and/or FVA without full Monte Carlo simulation. In our project, we implement the unilateral CVA engine for single instruments supporting IRS, Equity Swap and CCS, without Wrong Way Risk. In case that the the portfolio contains a single instrument and no Wrong Way Risk is required, we can use analytic or synthetic solution to get the unilateral CVA efficiently. The method we compute the unilateral CVA is from [Damiano Brigo and Pallavicini 2013].

5. IMPLEMENTATION

In this section, we discuss some high-level details of our project implementation, including some traps and pitfalls that we encountered during the implementation process.

5.1. Libraries and Frameworks

The computing engine, which is most fundamental part, is based on the QuantLib[QuantLib 2015]. We try to make our implementation conform with the other parts of QuantLib so it can be integrated. The compute engine exposes its feature using ASP.NET MVC WebAPI framework to create RESTful services via HTTP protocol.

The mobile application is created using Xamarin[Xamarin 2015] framework for supporting different platforms. Although at current stage we only support Android, we do have plan to bring it to other major platforms supported by Xamarin, like iOS and Windows Phone. The mobile application let the users input the data necessary for do the computation, then it calls the compute engine via the Web API interface, and when the compute engine finishes the task, the mobile application will show users the compute results (or the error message if the computation is failed).

5.2. Design of IRS BVA pricing engine

In the BVA Calculation under Monte Carlo simulation framework, we use defaultNPV method to calculate the NPV at default time τ, say $NPV(\tau)$. Only this method need be specified using on the corresponding instrument in our simulation, given default time τ and underlying processes as the parameters. However, when it comes to the BVA calculation of vanilla swap under Black model, the default NPV method is not compatible well.

Because under Black model, we treat $(NPV(\tau))^+$ and $(-NPV(\tau))^+$ as a swaption and do swaption pricing, then we could not simply use this method since the default information is not available for the method. In order to use the simulation framework for BVA calculation, we need add more parameters.

Fortunately, in the UCVA Calculation under Monte Carlo simulation framework, we can work around it exactly, since only the counterparty can default and the investor is risk-free under the Unilateral Default Assumption.

5.3. Stochastic short-rate yield term structure

We know that the formula of zero coupon bond price under short-rate model is

$$P(t,T) = E_t[-\mathrm{e}^{\int_t^T -r_u \mathrm{d}u}]$$

What's more, if there is analytic solution for above formula, we say the corresponding short-rate model is affine. Otherwise, we should calculate the solution using some numerical method.

$$P(t,T) = \mathrm{e}^{a(t,T)-b(t,T)r_t}$$

Since we only use one-factor affine model to describe short rate so far, we construct the stochastic short-rate yield term structure based on the analytic solution of zero coupon bond price.

5.4. Equit Swap

For the base swap class, it has a protected constructor, which can be used by derived classes to specify the number of legs of the swap. There is a Legs_ member, where we can placed all calculated cash flows. Normally, we should calculate all the future cash flow when constructing the swap.

5.5. AT1P Calibration

Under the AT1P model, the volatility depends on the time. When we use the BlackVol function of BlackScholesMertonProcess to return the volatility, we implicitly call the localvolatility function of the same class. This localvolatility function is used to calculate the time-depend volatility, contingent on the type of volatility term structure. In order for the result to be correct, we should take care of the input. Namely, the getVolTermStructure function of AT1P should return the monotone increasing variance structure for later calculation. Also the interpolation method used in the base class defaultmodel may affect the result of the AT1P model calibration, even though the difference is not wild.

6. TESTING

The project will have three levels of testing C unit test, system/integration (combined) test, and acceptance test. The details for each level are addressed in the approach section. The estimated time line for this project is very aggressive (five (5) months), as such, any delays in the development process or in the installation and verification of the third party software could have significant effects on the test plan.

6.1. Software Risk Issues

There are several parts of the project that are not within the control of the Risk Analysis Tool application but have direct impacts on the process and must be checked as well.

— The computation engine depends on third-party data, which we use for cross validation and model calibration.
— The application service is deployed on Windows Azure cloud environment.
— The client application is based on the Xamarin framework, which enables the cross-platform development for mobile application.

6.2. Items Will and Will Not be Tested

The following is a list of areas to be focused on during testing of the application.

— Kou jump process
— Kou process calibrator
— AT1P default model
— AT1P model calibrator
— CIR process calibrator
— Pricing engine for Equity Swap
— Pricing engine for European option using Kou model
— Credit VaR engine for IRS, Equity Swap and FX Swap
— CVA and DVA engine for IRS, Equity Swap and FX Swap
— FVA engine for IRS, Equity Swap and FX Swap
— Android mobile application for calling the application service

The following items are implemented in third party libraries and framework. We assume they are correctly implemented unless we find bugs affecting us.

— Geometric Brownian Motion process
— CIR process
— Pricing engine using Black-Scholes model
— Risk-free pricing engine for IRS
— Risk-free pricing engine for FX Swap
— Term-structures
— Monte Carlo simulation
— Calibration framework
— Web API framework that handling Web service call
— Xamarin GUI framework for mobile application

6.3. Testing Levels

The testing for the will consist of unit, system/integration (combined) and acceptance test levels, plus the bug regression tests. It was hoped that there would be a full time independent person for system/integration testing, and involve target users for participating the acceptance testing. However, with constraints and time line established, most testing will be done by the development teams participation.

Unit testing will be done by the developer. Proof of unit testing (test case list, sample output, data printouts, and defect information) must be provided by the programmer All unit tests must be automated.

System/integration testing will be performed by the development team. No specific test tools are available for this project. Programs will enter into System/Integration test after all critical defects have been corrected. A program may have defects as long as they do not impede testing of the program (i.e., there is a work around for the error).

Acceptance testing will be performed by the development team in users perspective and by volunteering class peers from other groups. The acceptance test will be done in parallel with the development process for a period of two weeks after completion of the system/integration test process. The application will enter alpha state and enter into acceptance test after all critical and major defects have been corrected. A program may have minor defect as long as it does not impede the functionality of the program (i.e., there is a work around for the error). Prior to final completion and sign-off of acceptance testing all open critical and major defects must be corrected and verified. After the sign-off, the application will enter beta state for public feedbacks.

In addition, in case of any bugs found in the application, an automated **bug regression test** must be created to reproduce the bug. After the bug has been fixed, the regression test case will be included in the unit test collection to prevent reintroducing the bug again.

6.4. Unit Testing

The granularity of the unit testing is functions of each class developed by the development team. The unit test collection must cover 100% of the functions and 80% of the code lines. For each function, there should be test cases scenarios shown below:

Normal case: for all functions, with normal input, the output of the function is correct. A core function should usually be covered by unit tests with all typical scenarios. The expected output should come from third-party source (e.g., research papers, books, etc.). If third-party source is not available, the expected output should come from independent form the spreadsheet calculation.

Reduced case: if the input that reduce the case to a simplified case which has wellknown results, the function will output the well-known result.

Extreme case: for functions with public interface, with extreme input value, the output of the function is consistent. Error case: for functions with public interface, with error input, the function should give error message

6.5. System/Integration Testing

The system/integration test verify the correctness of the application in an end-to-end approach. The granularity of the test case is the use case. For each feature, system/integration testing should cover below scenarios:

Normal scenario: when a typical input is given, the correct output should be provided by the application.

Error case: when the input is incorrect, the error message should be provided indicating the error. The application must not crash in error case.

Extreme scenario: depending on the requirement of the use case, treat it either as normal input or error input.

6.6. Acceptance Testing

The acceptance testing tests the features and use cases in end-users perspective. It covers similar scenarios as system/integration testing.

Besides correctness, the acceptance testing also include a reasonably satisfactory user experience, i.e., for most users with necessary domain knowledge, they should be able to use the application with reasonable degree of training.

REFERENCES

Massimo Morini Damiano Brigo and Andrea Pallavicini. 2013. *Counterparty Credit Risk, Collateral and Funding - With Pricing Cases for All Asset Classes*. John Wiley & Sons Ltd.

QuantLib 2015. QuantLib Official Website. (2015). http://quantlib.org/index.shtml

Hidde Schwietert. 2014. *Funding Value Adjustment and Valuing Interest Rate Swaps*. Master's thesis. University of Amsterdam.

Xamarin 2015. Xamarin Official Website. (2015). http://xamarin.com/