A Combination of V Development Life Cycle and Model-based Testing to Deal with Software System Evolution Issues

Imane Essebaa and Salima Chantit
Computer Laboratory of Mohammedia (LIM), Faculty of Sciences and Technics Mohammedia, Hassan II University of Casablanca, Mohammedia, Morocco

Keywords: Model-based Testing, V Development Life Cycle, Model Driven Development, Evolutionary System.

Abstract: Manage Testing is one of the most important and difficult phases in the development process. Indeed, it is a decisive step before deploying a product which aims to verify and validate whether the developed product satisfies customer requirements. This step becomes more and more difficult when we face evolutionary system requirement. Several works and approaches were proposed to deal with this issue but they do not describe any approach to well manage test phase in evolutionary software system. It is in this context that this paper proposes a new approach that resolves the problem of managing tests even in a system with evolutionary requirements. As our works are focused on Modeling Driven Development represented by its variant Model Driven Architecture, we will focus in this work on automating test generation from well known models as Model-Based Testing combined with development life cycle to deal with system evolutions. For this first work, we focus on V life cycle since it specifies different types of test needed to well test a system. In order to illustrate concepts used in our approach, we will present their application in a RentalCarAgency applications.

1 INTRODUCTION

Automating Development Process is become an important domain that several software companies are trying to adopt and apply in the realization of software product then to test it. Testing a system is considered as the most important and costly step in the development process as it is a decisive phase to validate if the system satisfies all customer requirements. Thus, automating this phase becomes essential.

Test step becomes more and more difficult where the system evolves. Indeed the main difficulty in software engineering evolution is that the customer requirements are able to change while software is being developed, thus dealing with continuous requirements evolutions and changing system is become the motivation to change the way how to make software products.

In this context several approaches and techniques have appeared in order to satisfy this motivation and evolve development process. In this paper we focus on two new areas that evolves rapidly and that are recently used by several software companies; Model Driven Engineering and Incremental Development process

As the subject of this paper is testing we limit our work to the corresponding variant of MDE which is Model-Based Testing. This method proposes to generate automatically tests from models which allow it to deal with changes.

At its turn, Incremental Development aims to develop the system incrementally and by interacting with the customer after the completion of each increment. In this work we chose V life cycle as one of the most used increments in the incremental development process.

We note that both, MBT and Incremental development aim to fluently manage frequent requirements changes; the incremental development focus on a methodological aspects that defines the process to develop and test the system while MBT is more concerned by an architectural aspect that aims to automatically generate test cases from requirements models.

This paper is organized as follow, in the second section we summarize concepts elaborated in this paper. In the third section we present the problematic that will be discussed and resolved in this paper following by a presentation and a discussion of previous works made in this context in the fourth section. The following section (Section 5) describes our proposed approach that will be illustrated in the sixth section on

Essebaa, I. and Chantit, S.
A Combination of V Development Life Cycle and Model-based Testing to Deal with Software System Evolution Issues.
DOI: 10.5220/0006657805280535
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 528-535
ISBN: 978-989-758-283-7

a case study of RentalCarAgency, and we finish by a conclusion and some of our future works.

2 OVERVIEW OF CONTEXT

2.1 Evolutionary System

Software evolution refers to the process of developing a software initially then repeatedly updating its requirements for various reasons. Indeed, software systems are bound to adapt or else "they become progressively less satisfactory in use" (Lehman and al, 2001)

The evolution of requirements in a software system may be omission, addition and/or modification in the specifications.

Requirements changing issues engender a very important problem which is managing testing phase. In this context, several approaches have appeared to solve this problem among which we highlight the Model-Based Testing approach

In this paper, we focus on how to manage testing in an evolutionary software system proposing an approach using Model-Based Testing with V development life cycle.

2.2 Model-based Testing

Testing a system is an activity performed to identify software problems and failures in order to improve the quality of a program.

The Model-Based Testing (MBT) is a variant of test techniques that are based on explicit behaviour models, describing the expected behaviours of the System Under Test (SUT), or the behaviour of its environment, built from functional requirements. The MBT is an evolutionary approach that aims to generate automatically from models, test cases to apply on the SUT (Utting and al, 2007).

The figure 1 represents a standard process of Model-based Testing where the process of designing models and the process of developing the System Under Test (SUT) evolute in the same time.

2.3 Evolution in Model-based Testing process

Based on the figure 1 we can define which entities can evolve when of system Evolution:

- Requirements.
- The test model.

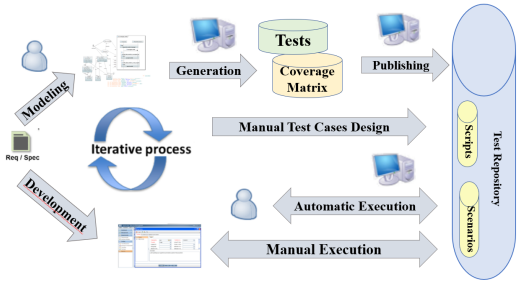

Figure 1: Model-Based Testing process.

- Test cases.

The evolution in these elements engenders modifications in some entities in MBT process:

- The cover matrix
- The test repository
- Executable scripts

In their paper (Bouquet and al, 2011) Bouquet and al. admit that each evolution of the requirements is described informally in the specification then the test engineer performs this change in the test model, which is sent to the testing builder. The latter propagates the evolution in the cover matrix and in the generated tests. The export of these tests modifies the test repository. Finally, the scripts are updated to allow their execution on the SUT. This way the process can allow testing from models of scalable systems.

2.4 V Life Cycle

In this first work on the testing domain, we will focus on V development life cycle. We chose V model because of its flexibility to return to the previous step if any problem is detected.

In the V model development life cycle, based on the same information the development and test activities start. Indeed based on requirements document developer team start analyzing and working on the design phase. After completion of this step, developer team starts the implementation phase and a testing team starts working on test planning, test case, and test scripting. Activities in V model are working parallel to each other.

Typical V-model shows Software Development activities on the Left hand side of the model and the Right hand side of the model describes actual Testing phases that can be performed:

- Unit testing
- Integration testing
- System testing
- Acceptance testing

3 PROBLEMATIC

By analyzing Model-Based Testing process and V development life cycle we note that dealing with tests in system with requirements evolution is a common challenge that both these domains aim to resolve. However MBT and V life cycle evolve separately and their combination should be taken into consideration. Using V life cycle and MBT in one approach to resolve system evolution issue necessitates a response to the following questions:

- How to manage the evolution of the system using models in a V process?
- To what extent is it possible to automate testing process taking into consideration system evolving?
- Which models should we use to generate test cases?

4 RELATED WORKS

Being aware of the importance of incremental development process and Model-Based Testing, many works were made on these domains in order to improve development process taking into account managing system changes. However we note that these domains evolve separately and their combination was discussed in few works that are presented in the following of this section.

In their paper (Bouquet and al, 2011), the authors present an implementation tool using Model-Based Testing that deal with system evolutions. In their work, they consider different test suites to test SUT after evolution (Evolution, stagnation, regression and deletion). For each test suite, they propose a rule to define it. For the generation of these test suites, they use TestDesigner which is based on Class Diagram and State Machine Diagram with OCL constraints. Using OCL constraints to generate tests does not allow to validate all the system , the paper does not describe how to define a set of test to apply to SUT after evolution. The the basic idea in this paper is to ensure the preservation of security properties for long-living evolving systems using software testing.

Pretschner and al.present in their paper (Pretschner and al, 2001) the importance to deal with evolution of systems in testing phases. They also present their AutoFocus tool which implement Model-Based Testing and that aim to generate test cases. The paper describes some evolution development process. However, this paper does not propose any method or approach to how system requirements should be modeled neither how AutoFocus generate test cases and from which model. We note also that this paper does not explain how to deal with evolution of system using Model-Based Testing.

In the paper (Blackburn and al, 2005), the authors discuss how organizations use specific model-based tools and evolved their existing engineering processes to develop and test applications. The paper highlights challenges and best practices of Model-Based Testing and its integration in developments life cycles, however it does not present which model to use to generate tests.

In their work (Güldali and al, 2010), Güldali and al. discuss how Model Based Testing can support Agile Development without conflicting with the principles of Agile manifesto. According to their discussion, MBT fits very well with agility, indeed the main advantage of MBT for the agile world is the usage of models as primary artifacts and the automation of several test activities. This paper is an introduction of other works where they propose an approach to combine MBT with scrum methodology (Löffler and al, 2010).

In their paper (Löffler and al, 2010), authors propose an approach based on models to improve customer specifications and acceptance testing in scrum. In their approach, authors use UML models likeInteraction Overview Diagram (IOD) and Sequence Diagram (SD) to represent user stories specifications. These diagrams are used by both developers and testers. Testers enhance them by Test data to generate automatically test tables, which can be executed by selenium or fitness tools, this approach is an early stage and focused only on acceptance testing, therefore the paper does not describe how to manage other testing phases(Unit testing, Integration testing and System Testing).

For the two last presented works in this section, we note that the use of scrum methodology does not allow to test all system in its different phases, but only the validation step to confirm if the developed system satisfies all customer requirements.

Most of works that are based on agile methodologies especially scrum combine it with a development life cycle in each iteration (Such as V or Y life cycle). In order to well manage all development phases, in next sections of this paper, we will focus on V life cycle and we will show how to combine it with Model Driven Development and Model-Based Testing.

5 PROPOSED APPROACH

In this section we present our approach to combine Model-Based Testing and Agile Development in order to manage evolution of software system's requirements.

In the first part of this section we describe our approach how to manage testing in different types of system evolutions. In the second part we describe the integration of MBT in V life cycle. In the last part of this section we present our approach of managing evolution in the combined approach of V process and MBT.

5.1 Managing Test in Evolutionary System

Applications that evolve must be validated before they are redeployed on the market, especially if they are critical ones. Managing tests in evolutionary systems is one of the primordial area studied by researchers. The validation process in the previous approaches takes into account the test of the code before requirements evolutions.Then the process ensures that the changes in the code, due to the evolution of the requirements described in specifications, have no impact to the code. This process is called non-regression tests. Non-regression test can be useful at different levels of the test: unit, integration, system or acceptance.

The purpose of the non-regression test approaches is to select tests to verify the correct operation of the unchanged parts in the System Under Test after modification. However, there is no equivalent to dealing with validating the deletion of the system features that have become evident in the evolution process, or of testing the new features.

In this work, we define three types of system' evolution; Modification or deletion of existing features and addition of new features. In our approach we consider that these types may happen in one evolution.

To well explain our approach, we consider a first system F that contains several features $f_i : F = \{f_i\}$. To test this system we generate a set of tests $T=\{t_j\}$ composed from several tests t_j to validate the system requirements at each level of the development process.

After an evolution, we get a new system F' with new features: $F'=\{f'_i\}$. To test this system we use at first non-regression tests to test the unchangeable part of the system, and at second step we generate another set of test $T'=\{t'_j\}$ which is an union of different set of tests corresponding to each evolution type, so T' in our approach is $T' = (T_{nr} \cup T_m \cup T_a \cup T_d)$ where:

- T_{nr} : *Non-regression tests to that validate unchangeable features in the system.*
- T_m: *New generated tests of modified features, this set can be empty if there are no modifications in features.*
- T_a: *Tests of new features added to the new system, if there is no addition to the system Ta is empty.*
- T_d: *Tests of feature deletion from the old system, to test if features are really deleted from the new system that can be empty if there are no features deleted .*

5.2 Approach of Integration Model-based Testing in V Life Cycle based on Model Driven Architecture

In this section, first we present our approach to model system requirements using MDA approach integrated with V life cycle and their combination with MBT, the second part is to present some rules that we aim to use in order to automate test generation.

Modeling System Requirements Applying MDA Approach:

To well manage requirements evolution of the system, we aim to combine V process and MBT at first work, our approach consists on:

- Cover Requirements and functional specifications steps in V life cycle by CIM level of MDA which is represented in our approach by SBVR and UCD, we use these diagrams for both generate "Validation tests" to validate if the developed system responds to described requirements.
- Generate the High level design represented in our approach by PIM level which is generated automatically from CIM level (To generate PIM level form CIM one we use our approach defined in our previous works (Essebaa and al, 2017)), this step is represented by BCD and SSD of each use case element, we generate "Integration tests" from these diagrams to test the correct functioning between different elements of the system.
- Generate the low level design represented by PSM level which is modeled by CD and DSD (The approach we propose to automate transformations between PIM and PSM levels that will be discussed in our future works). We generate "Unit tests" from this level to test the generated code.

Test Generation Rules:

Rule 1: Generate Validation Tests from CIM Level

In our approach, Validation tests are generated from different fact types (is_property_of and associative) of SBVR standard that details system requirements presented in Use Case Diagram. To validate each system feature, we use business rules that describe relationship between different fact types of the system (Possibility, Obligation, Necessity, ...).

Rule 2: Generate Integration Tests from PIM Level

In our approach we tend to generate Integration tests from PIM level which is represented using Class Diagram and System Sequence Diagram. In this approach, we use System Sequence Diagram to extract the entities that participate to ensure the feature, then we test if the relationship between these entities responds to the requirement.

Rule 3: Generate Unit Tests from PSM Level

Unit tests in our approach are generated from PSM level that we aim to model using Detailed Sequence Diagram and Detailed Class Diagram. In this level we will test each operation in class diagram that participate to realize a feature, same to previous level the operation that should be tested is extracted from Detailed Sequence Diagram.

The table 1 bellow summarize these rules:

Table 1: Test generation rules.

Rule	Model	Target
SBVR&UCD2VT	Use Case element	Requirement to validate
	Fact Type	Sub feature to test
	Business rules of a Fact Type	Validation Tests
BCD&SSD2IT	Actor and data_object Life Lines	Classes to test
	Relationship between classes	Integration Tests
CD&DSD2UT	Messages	Operations to test
	Operation in classes	Unit test

5.3 Managing Evolution in a Combined Approach of V Process and MBT

In the two previous sections, we discussed how to manage all types of evolution in our approach and how we aim to combine V life cycle and MBT.

In this section we will present our approach of managing tests in evolutionary system inside V life cycle using MBT. As we have different types of tests clearly defined in V life cycle, our proposal consists on:

- Modeling the new CIM level after specifying the evolution, then generating new "acceptance tests" and "system tests".
- Generating the new PIM level, then generating new "Integration tests".
- Generating the PSM level, then "Unit tests" to test the new generated code.

The definition of the new set of tests is done by applying the rule previously defined in the first part of this section, however the approach of how to generate test cases from models will be defined in our future works.

The figure bellow describes the presented approach:

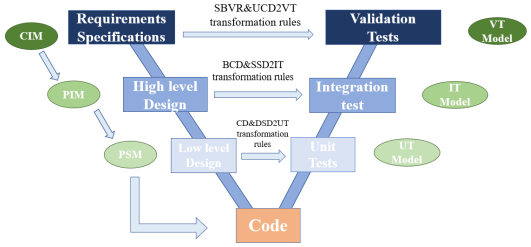

Figure 2: Combination of MDA, MBT and V life cycle.

The combination of MBT and MDA inside a V life cycle is to fluently manage the evolution of system' requirements, indeed incremental developments allows to communicate with the customer after each increment and test developed features.

The V life cycle well illustrate this approach by separating requirement specifications and their development from testing phase which are made in parallel.

6 CASE STUDY

6.1 General Description of RentalCarAgency System

In order to illustrate the different concepts introduced in this paper, we present in this section the case study of RentalCarAgency application proposed in our previous works (Essebaa and al, 2017).

At first we present initial requirements of the application to realize and their modeling applying our MDA approach, then we present test cases dispatched to test types in V life cycle.

In the second part of this section, we describe some evolutions in RentalCarAgency system in order to apply our approach of generating the new set of test cases to verify and validate the new system under test. The application have three users profiles that have different privileges:

- Customer: A person who can view the cars available in the agency, rates and promotions and may

subscribe. Once registered, the visitor becomes a client of the Agency. A client must authenticate in the system to search for available cars and book a car by indicating the reservation date and time.

- Manager: A Manager must also authenticate to view all cars, add, edit or remove cars and view the bookings made by customers waiting for validation to decide to accept or decline them.
- Administrator: Once authenticated into the system, the administrator has the privilege of modifying and deleting a customer account, as well as the management of managers account (add, change or delete).

We can define some management rules as below:

- A customer can rent several cars.
- A car can be rented by 0 or several customers.
- A manager can manage 1 or more cars.
- A car is managed by 1 or more managers.
- An administrator can manage 1 or several customer accounts.
- An administrator can manage 1 or more account managers.

The first step in our approach is to model the system requirements following our approach as described in the previous section of this paper:

- **Modeling CIM Level and Generating Validation Tests:** In this step we model the system requirements using SBVR (Semantic Business Vocabulary and Business Rules) and UCD (Use Case Diagram) to represent the CIM level, from where we tend to generate Validation tests.

From this level we will generate validation tests. The table 2 bellow describes the application of validation test generation rule on RentalCarAgency system (we will focus in this part on Use Cases elements specified for "Customer" actor).

- **Modeling PIM Level and Generating Integration Tests:** The second step consists on generating PIM level represented by BCD (Business Class Diagram) and SSD (System Sequence Diagram) and then generating Integration Tests from PIM level, the table 3 describes the application of this rule on RentalCarAgency system.

  The generation of PIM level was automated using an eclipse plugin that we developed and presented in our previous works (Essebaa and al, 2017)

- **Modeling PSM Level and Generating Unit Tests:** The last step consists on generating PSM level that we aim to model it by CD (Class Diagram) and DSD (Detailed Sequence Diagram) ansd then generating Unit Tests from PSM level, the table 4 describes the application of this rule on Rental_Car_Agency system.

  The generation of PSM level will be discussed and detailed in our future works

- **Evolution of Rental_Car_agency system**

  In this section we will make some evolutions to the system (addition, deletion and modification of features) in order to visualize generation tests attitude in evolutionary system.

  – *Modifications:* "View car_catalog" feature will be available for all users not only customers, this modification engender a new actor "User" that it will be a generalization of "Customer" actor.

    The modification of this feature will necessitate to make some modification to test cases. Indeed as in our method we use MBT approach we tend to generate automatically the new set of tests according to the new models after modification, we call these tests T_m for "Tests of modification" At this stage of work we propose a manual definition of the tests affected because of the change, the table 5 describes these tests

  – *Addition:* In the new system, "Customer" will be able to validate its rental by payment, "payment" feature will generate new test cases that we will call T_a for "Tests of addition". The addition of a feature may engender some modifications in old ones, in table 5 we define new tests added to test payment feature and describes if other tests was affected due to addition.

  – *Deletion:* The addition of "payment" feature requires to delete "manage rental" feature of "Manager" that allowed him to accept or reject the rental, in the new system the customer can validate its rental from the system, before proceeding to payment option the system must be able to check if the chosen car is available for date specified by the customer. We note that deleting feature may also add or modify tests, otherwise tests corresponding to deleted feature that we represent T_d, the table 5 describes these tests.

Definition of Set of Tests of the New System Under Test After Evolution: After defining evolution and generating new tests for each evolution, we have to define the new set of tests that we will apply on the

Table 2: Generation of Validation tests form CIM level.

Source			Target		
UseCase Element	*Fact Type*	*Business Rule*	*Requirement*	*Sub_feature*	*Test cases*
View car_catalog	Customer requests car_catalog	It is possible that customer requests car_catalog if customer is_authenticated	The system must allow authenticated customer to view car_catalog	request car_catalog	Authenticated customer must have the possibility to request car_catalog
	Customer receives car_catalog	It is possible that customer receives car_catalog if customer is_authenticated		receives car_catalog	The system must sends a car_catalog if it was requested by a customer
	Customer views cat_catalog	It is possible that customer views car_catalog if customer is_authenticated		view car_catalog	The customer must be able to view recieved car_catalog
books car	customer books car	It is possible that customer books car if customer is_authenticated	The system must allow authenticated customer to books car	books car	Authenticated customer must be able to books car
	system creates rental data_object	It is obligatory that system generates rental if customer books car		generates rental data_object	The system must generates the rental data_object for the booked car
logs_into	system requests user_credential	It is obligatory that the system requests user_credential if customer logs_into system	The system must allow customer to log_into the system	requests user_credentail	The system must request user_credential if a customer try to logs_into the system
	customer sends user_credentials	It is necessary that customer sends user_credentials if system requests user_credential		sends use_credentials	The system must allow customer to send user_credential
	customer sends user_credentials	It is necessary that customer sends user_credentials if system requests user_credential		sends use_credentials	The system must allow customer to send user_credential
	System verifies user_credentials	It is obligatory that the system verifies user_credentials if customer sends user_credential		verifies user_credential	The system must be able to check user_credential
	System accepts user_credentials	It is possible that system accepts user_credential		accepts user_credential	The system must be able to accept user_credential

Table 3: Integration tests generation from PIM level.

Source		Target	
Requirements	*SD connection*	*Classes*	*Integration tests*
Books car	The operation requires connection between "Customer", "Rental" and "Car"	Customer	Customer books at least 1 car
		Car	Car belongs to 1 rental
		Rental	Rental contains at least 1 car
Logs_into	The operation requires connection between "Customer" and "Account"	Customer	Customer owns 1 account
		Account	Account belongs to 1customer

Table 4: Unit tests generation from PSM level.

Source			Target
Requirements	*Messages of SD*	*Operations to test*	*Unit tests*
Logs_into	System requests User_Credential	requests(user_credential)	Test "requests" operation
	Customer sends User_credential	sends(user_credential)	Tests if user_credential are not empty
	System verifies User_credential	verifies(user_credential)	Test if user_credential are correct
	System accepts user_credential	accepts(user_credential)	accepts if user_credential are correct
	System rejects user_credential	rejects(user_credential)	rejects if user_credential are not correct

new SUT, in our approach to define this set we have to apply the rule previously defined! T'= ($T_{nr} \cup T_m \cup T_a \cup T_d$),where T_{nr}: all test cases that did not evolute with the evolution of the system.

Table 5: Evolution of tests.

Type	Requirements	Tests		Evolution
T_m	View car_catalog	VT	Every user must be able to view car catalog	Change
		IT	Coordination between "User" and "Catalog"	
		UT	Test view(catalog) method	
T_a	Payment	VT	Customer must be able to pay its reservation	Addition
			The system must verify Credit card and code	
		IT	Coordination between "Customer", "Rental", "Car" and "Payment"	
		UT	Test pay(rental, customer) method	
T_d	Manage rental	VT	Manager accepted rental Manager rejected rental	Deletion
		IT	Coordination between "Manager", "Car", "Rental"	
		UT	Test manage(rental, manager)	
T-a	Validate date	VT	The system must be able to verify automaticaly the availibility of car in the chosen dates	Addition
		IT	coordination between "Car" and "Rental"	
		UT	Test validate(date, car) method	

7 CONCLUSION

The primary objectives of this paper is to present our approach of combining two important variants of Model Driven Engineering; MDA and MBT, and V life cycle to manage testing in an evolutionary software system.

Indeed in this first work we chose to combine these process to deal with system changing issues, we define in this paper three types of software system evolution; modification or suppression of existing features, and addition of new features.

In this paper we also present a rule that allows to define a set of tests after evolutions, this rule is a unions of different sub-test corresponding to each type of features previously defined and old tests of unchanging part of the old system.

We mention that this paper in an introduction of our future works where we aim to improve the idea of testing in evolutionary system using Model-Based Testing.

8 PERSPECTIVES

In our future works we plan to :

- Integrate MDA and MBT in the most widely used development methodologies even agile ones.
- Create an eclipse plugin (In the continuity of our previous works)(Essebaa and al, 2017), that automatically generate tests form models.
- Apply our approach on more consistent projects in order to evaluate our approaches and test tools.

REFERENCES

Blackburn and al (2005). Life cycle integration of model-based testing tools.

Bouquet and al (2011). A model-based testing approach for evolution.

Essebaa and al (2017). Tool support to automate transformations between cim and pim levels. In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: MDI4SE,*, pages 367–378. INSTICC, SciTePress.

Güldali and al (2010). Agility vs. model-based testing: A fair play? In Bode, S. e. a., editor, *Proceedings of the IWK2010 Workshops: The First International Workshop on Evolution Support for Model-Based Development and Testing (EMDT2010)*, volume 646 of *CEUR Workshop Proceedings*, pages 55–58. (invited paper).

Lehman and al (2001). Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 11(1):15–44.

Löffler and al (2010). Towards model-based acceptance testing for scrum. *Softwaretechnik-Trends*, 30(3).

Pretschner and al (2001). Model based testing in evolutionary software development.

Utting and al (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.