Matching Antipatterns to Improve the Quality of Use Case Models

陳素清
951127

Reference

- Matching Antipatterns to Improve the Quality of Use Case Models
  - Mohamed El-Attar and James Miller
  - 14th IEEE International Requirements Engineering Conference (RE'06)
- http://www.steam.ualberta.ca/main/research_areas/ARBIUM.htm
- http://www.steam.ualberta.ca/main/research_areas/Use%20Case%20Antipatterns%20Website.htm

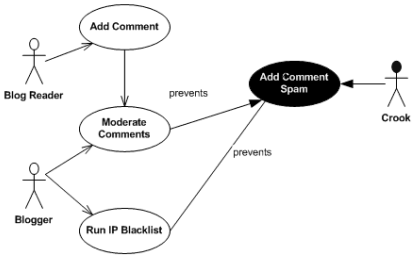
Outline

- Introduction
- Related Work on Improving the Quality of UC Models
- Antipatterns of UC Modeling: A New Approach to Improving the Quality of UC Models
- Automated Risk-Based Inspection of UC Models (ARBIUM)
- The MAPSTEDI System Case Study
- Conclusions

Introduction

--document those “bad practices” in the form of antipatterns

- UML: standard of producing software blueprints
- Use Case Modeling: method to capture a system’s functional requirements.
- UC models
  - components constructed using natural langrage
  - diagrams that contain a small notational set
  - adhere to few syntactic rules.



- **Little Guidance is provided on constructing semantically correct UC diagrams.**
  - A knowledge management mechanism will document previous experiences and knowledge of senior analysts on potentially hazardous techniques and decisions, or bad practices.

Outline

- Introduction
- **Related Work on Improving the Quality of UC Models**
- Antipatterns of UC Modeling: A New Approach to Improving the Quality of UC Models
- Automated Risk-Based Inspection of UC Models (ARBIUM)
- The MAPSTEDI System Case Study
- Conclusions

Related Work on Improving the Quality of UC Models

- (1) **Guidelines** (2)**Refactoring** (3) **Templates**
- (4) **Verification**
  - Susan: uses symbolic model checking, must input a vast number of values regarding actors, UCs, and conditions, which is usually unavailable until the end of a project.

- **ARBIUM**
  - a set of rules to apply to any UC model to improve
  - A simple framework that designers can code additional rules

Quality Aspects of UC Models

Quality Attribute	Definition	Severity of its absence
Correctness	The UC diagram and textual descriptions must correctly represent the underlying requirements.	Very severe – Designers may build a faulty system.
Consistency	The textual descriptions must conform to the concepts and requirements encoded in the diagram and vice versa.	Very severe – Designers may build a faulty system.
Analytical	The model should not contain any design or implementation decisions, including interface details.	Variable severity – Designers'creativity is restricted. Moreover, it obscures the purpose of the system.
Understandability	The model must be readable and unambiguous. All stakeholders must share a common understanding of the presented functional requirements.	Variable severity – Can be severe if designers cannot understand the model or misunderstand its requirements.

Why is it Difficult to Produce High Quality UC Models After All?

- **UC Descriptions are Informal**
  - natural langrage
- **Iterative Development**
  - UC models become more complete with each phase.
- **UCs Present Only a Subset of a System's Requirements**
  - Non-functional requirement are outline in a supplementary specifications document.
- **Confusion about the UC Modeling Syntax**
  - 2003 "Errors in the UML Metamodel?": 25 errors in the UML2.0 metamodel of UC modeling.
- **Domain Expertise**
  - Safety critical power plant systems vs distributed bank systems

Outline

- Introduction
- Related Work on Improving the Quality of UC Models
- **Antipatterns of UC Modeling: A New Approach to Improving the Quality of UC Models**
- Automated Risk-Based Inspection of UC Models (ARBIUM)
- The MAPSTEDI System Case Study
- Conclusions

Benefits of Antipatterns

- An antipattern is a method shows users
  - how to arrive at a good solution from a fallacious solution and how to avoid the specious solution.

Name: The title of the antipattern.
Description: the unsound diagrammatic structures's des.
Rationale: A list of the deceptive reasons as to why the "Fallacious solution" seemed to be the right thing to do.
Consequences: A list of the harmful consequences.
Detection: Where: A guide to the areas where the antipattern can exist. How: Instructions used to detect a match for the antipattern.
Improvement: Actions that can convert a "Fallacious solution" into a better one or avoid the "Fallacious solution".

The Two Dimensions of Antipatterns

Situation	Tool Support Available	No Tool Support Available
Domain-independent Antipatterns	(1)	(2)
Domain Dependent Antipatterns	(3)	(4)

(1) which this paper focuses on, analysts are provided with a large number of domain-independent antipatterns.

(2)analysts are provided with a number of domain-independent antipatterns, also available

(3) the antipatterns must be developed by the domain experts. Using the simplified UC metamodel, analysts can construct OCL statements that accurately describe their antipatterns.

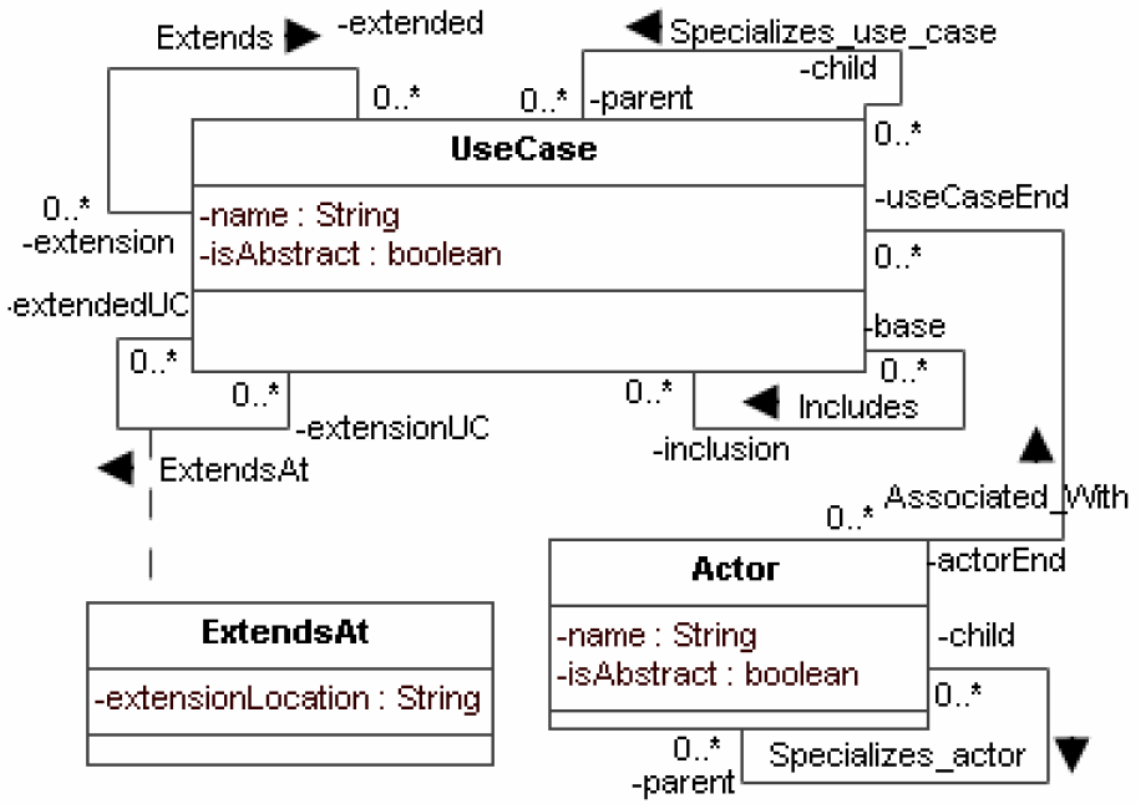
(4) analysts still need to develop their own antipatterns.

Describing Unsound Structures Using OCL

- OCL
  - be mostly used to describe a set of constraint that need to be satisfied in order to confirm that an antipattern did not match.
  - be used to describe constraints and invariants for object structures.
- Therefore, UC diagrams need to be transformed into object models.
- It is possible since every instance of a UC diagram conforms to the UC metamodel put forward by OMG.

The simplified version of the UC metamodel used in ARBIUM

- UCs
- actors
- include relationship
- extend relationship
- association relationship
- generalization relationship



All syntax rules are expressed with OCL statements

Examples of UC Modeling Antipatterns

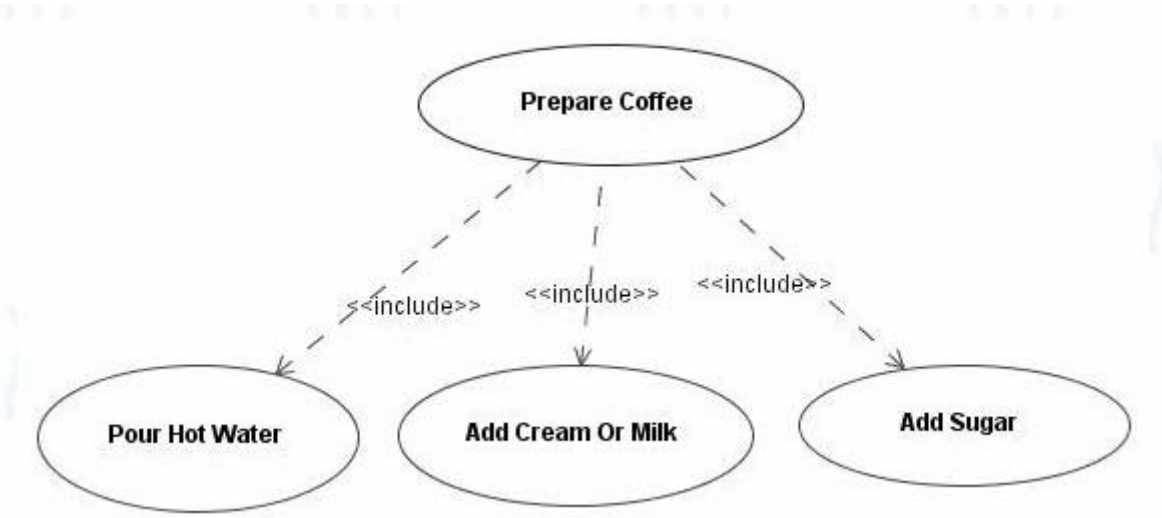
Name
Description
Rationale
Consequences
Detection
Improvement

- **Antipattern Name**:
  - Functional decomposition of UCs
- **Description**:
  - Modelers may divide the functionality of UCs over a number of UCs as means of increasing modularity.
  - Therefore, “smaller” UCs are created to contain portions of the behavior that is supposed to be carried out by a single base UC.
  - This concept of modularity may be applied in three forms:

Functional decomposition of UCs(Description)
--Using the include relationship

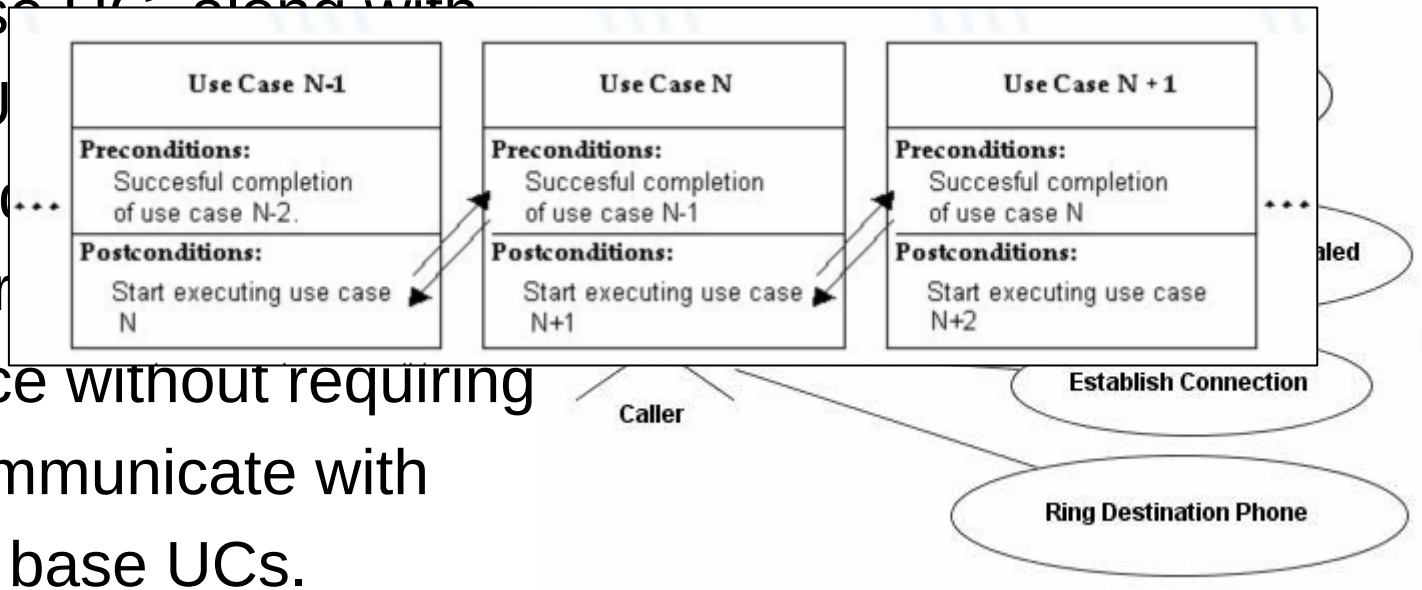
- Modelers create included UCs to contain portions of a complete service offered by a base UC.
- It is evident that the three inclusion UCs need to be all executed in order to fulfill the main goal of preparing a cup of coffee.

Name
Description
Rationale
Consequences
Detection
Improvement



Functional decomposition of UCs(Description)
--Sequencing UCs using pre and postconditions

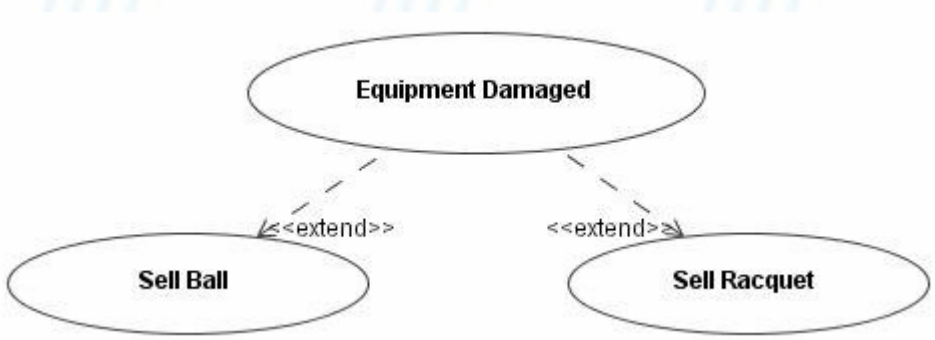
- When a complete service offered by a base UC is subdivided into “smaller” base UCs, it is often the case that the “smaller” UCs need to be performed in a particular sequence to properly execute the complete service.
- A bas… any U… shoul… perfor… service without requiring to communicate with other base UCs.



Functional decomposition of UCs(Description) --Using the extend relationship

- the improper use of the extend relationship.
  - An extension UCs are modeled to have knowledge of the base UC.
- If an extension UC contains general behavior that would be useful to more than one base UC, this would be a strong indication that the extension UC has degraded into a function.

Name
Description
Rationale
Consequences
Detection
Improvement



Examples of UC Modeling Antipatterns (Rationale and Consequences)

Name
Description
Rationale
Consequences
Detection
Improvement

- **Rationale**:
  - Since functionally decomposed UCs are naturally “smaller”, they will be easier to implement, understand, implement and test.
- **Consequences**:
  - It’s obscure the real purpose of the system.
  - Functionally decomposed UCs are often a first cut attempt into embodying design decisions.
  - These “smaller” UCs offer no value to the system’s users.
  - It’s may lead to more complex descriptions of the interactions between the actors and the UCs.

Examples of UC Modeling Antipatterns (Detection)

- Detection:
  - Using the include relationship:
    - search any *inclusion* UCs in the UC diagram.
    - The antipattern is matched if the inclusion UC is included only once.

Name
Description
Rationale
Consequences
Detection
Improvement

OCL Description:
context UseCase
inv **NotJustOneInclude**: not (self.base->size = 1)

Examples of UC Modeling Antipatterns (Detection)

- Sequencing UCs using pre and postconditions:
  - The analyst needs to examine the preconditions and postconditions of each base UC depicted in the UC diagram.

(1) a postcondition for a UC requires the initiation of another UC.

(2) a pre-condition of a UC requires the successful completion of another UC.

(3) a post-condition of a UC and a precondition of another UC, state similar requirements for a particular variable value.

Name
Description
Rationale
Consequences
Detection
Improvement

cannot be described using OCL

Examples of UC Modeling Antipatterns (Detection)

- Using the extend relationship:
  - The analyst needs to search for any extension UCs in the UC diagram.
  - If the extension UC extends more than one UC, the antipattern is matched.
  - check if it is too generic or specific?

Name
Description
Rationale
Consequences
Detection
Improvement

OCL Description:
context UseCase
inv ExtendingMoreThanOneUseCase:
not (self.extended->size +
self.extendedUC->size>1)

Examples of UC Modeling Antipatterns (Improvement)

Name
Description
Rationale
Consequences
Detection
Improvement

- Improvement
  - Offer
    user.

    system's

(1) the
shou

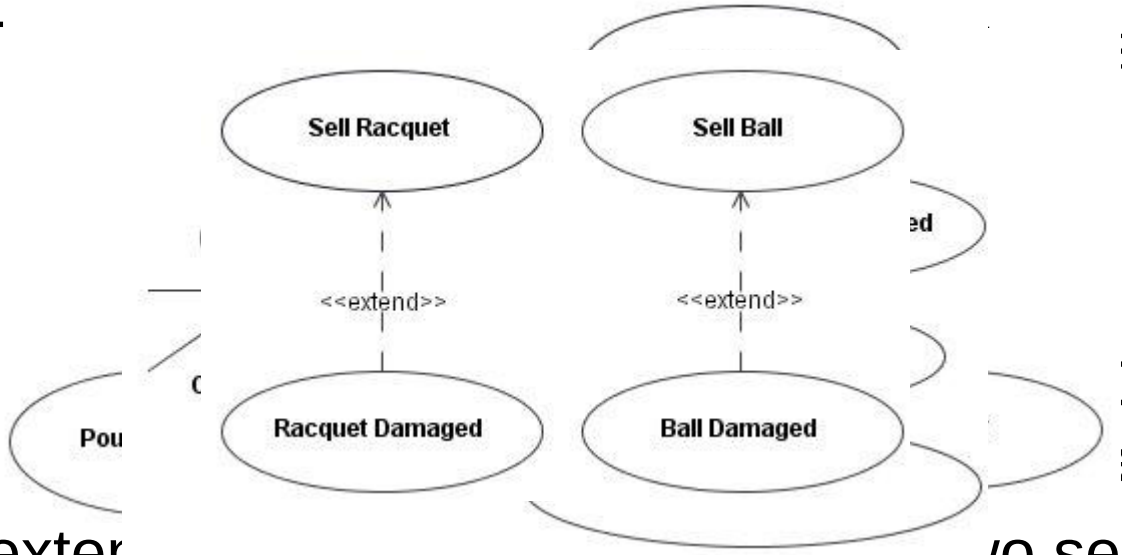
Cs
Coffee.

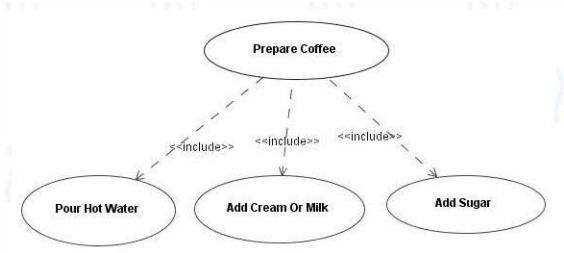
(2) The
be cc

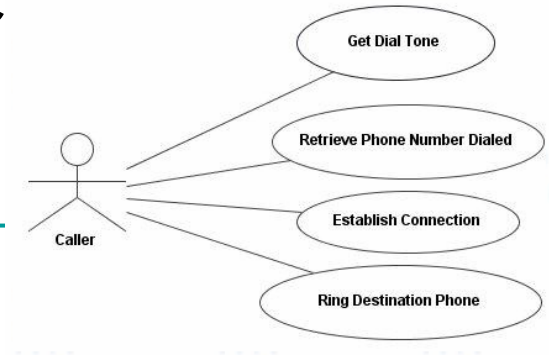
Cs should
e Call)

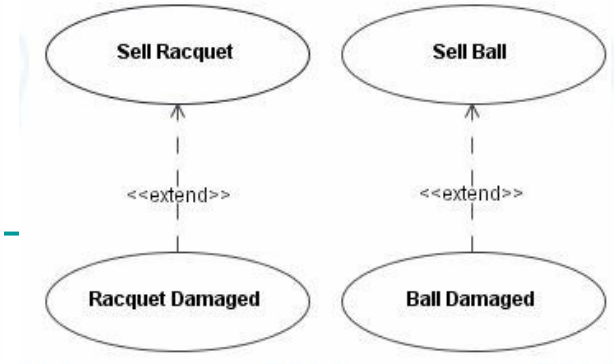
(3) the exten
extension UC

o separate









How to Use Antipatterns to Search For Risk Areas in UC Models

- Detailed guidelines
  - an analyst how to search a UC model that seem risky or debatable

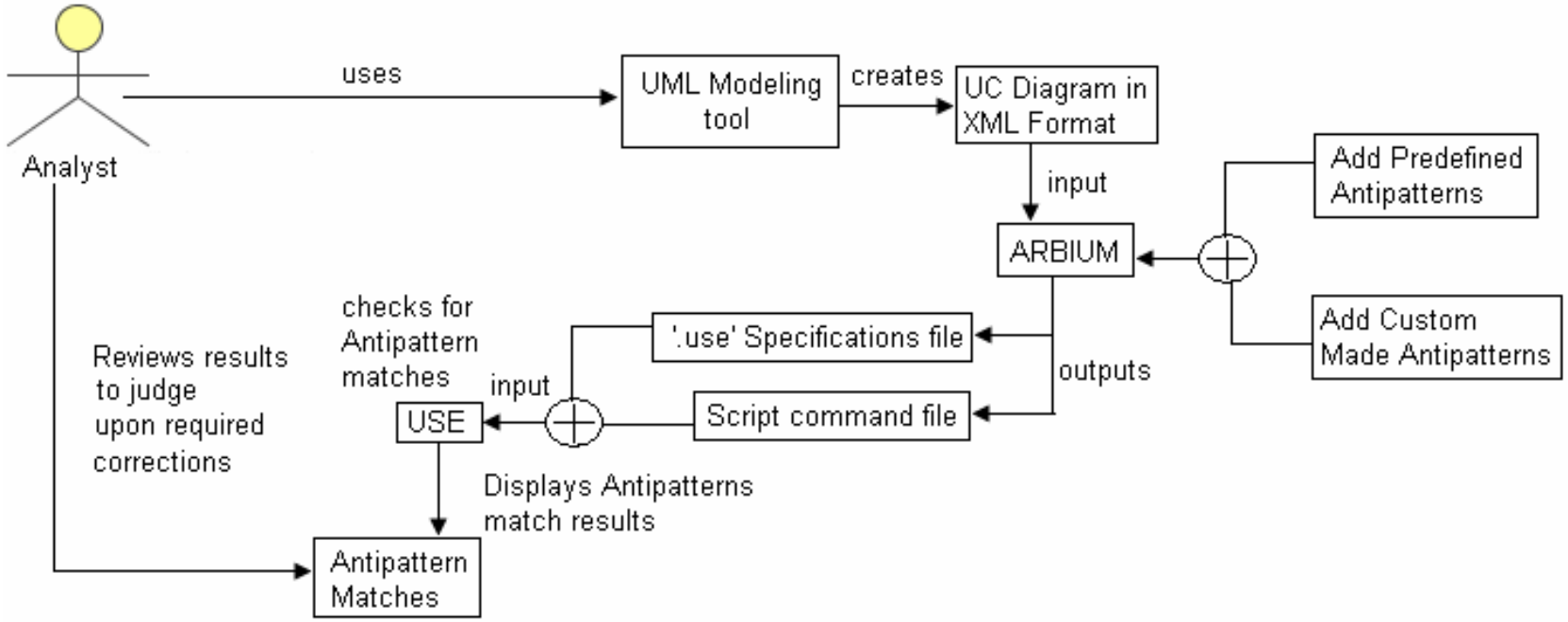
1. **Search**: If a UC model matches the description of an unsound structure, then that antipattern is matched,
2. **Judge**: the analyst is then required to judge whether or not corrective measures are required.
3. **Consequences**: changing the structure of the UC diagram may cause a new antipattern matches.

- Employed iteratively until all antipatterns are addressed.

Outline

- Introduction
- Related Work on Improving the Quality of UC Models
- Antipatterns of UC Modeling: A New Approach to Improving the Quality of UC Models
- **Automated Risk-Based Inspection of UC Models (ARBIUM)**
- The MAPSTEDI System Case Study
- Conclusions

Automated Risk-Based Inspection of UC Models (ARBIUM)



- **ARBIUM** targets the other type of deficiencies, those that require human intervention.
- **USE** is a system that checks the integrity of information systems against constraints described in OCL.

Outline

- Introduction
- Related Work on Improving the Quality of UC Models
- Antipatterns of UC Modeling: A New Approach to Improving the Quality of UC Models
- Automated Risk-Based Inspection of UC Models (ARBIUM)
- **The MAPSTEDI System Case Study**
- Conclusions

The MAPSTEDI System Case Study

- The MAPSTEDI system is a distributed biodiversity database system used to merge together the separate specimen collections.
- The order of applying antipatterns are irrelevant.

UC Diagram	UC count	Actor count	Illustrated Behavior
Database Access	4	2	Database accessibility right
Database Edits	3	3	Editing and updating the database
Database Queries	5	0	Provides a hierarchy of the query services offered by the system
Database Integrator	7	1	Integration of local and remote databases.
Administrative Process	3	2	Shows the administrative responsibilities

Antipattern matches in each iteration

	UC Diagram	Antipattern Matched	Quality Improved – Potential Severity
First Iteration Matches	Database Access	Multiple actors associated with a UC	Correctness: very severe
	Database Queries	Multiple actors associated with a UC	Correctness: very severe
	Database Iterator	Multiple actors associated with a UC	Analytical: non critical
		Multiple actors associated with a UC	Analytical: non critical
	Database Edits	Multiple actors associated with a UC	Analytical: non critical
		Multiple actors associated with a UC	Correctness and Consistency: very severe
	Administrative Process	Multiple actors associated with a UC	Correctness: very severe
		Multiple actors associated with a UC	Correctness: very severe
		Multiple actors associated with a UC	Correctness: very severe
Second Iteration Matches	Merged UC Diagram	Accessing a generalized concrete UC	Correctness: very severe
		UCs containing common and exceptional functionality	False positive

Conclusion

- Currently developing a large number of domainindependent antipatterns, which can be used to improve the quality of other types of UML models;
- ARBIUM facilitates the matching process and provides a framework for analysts to specify their customized antipatterns.

Question?

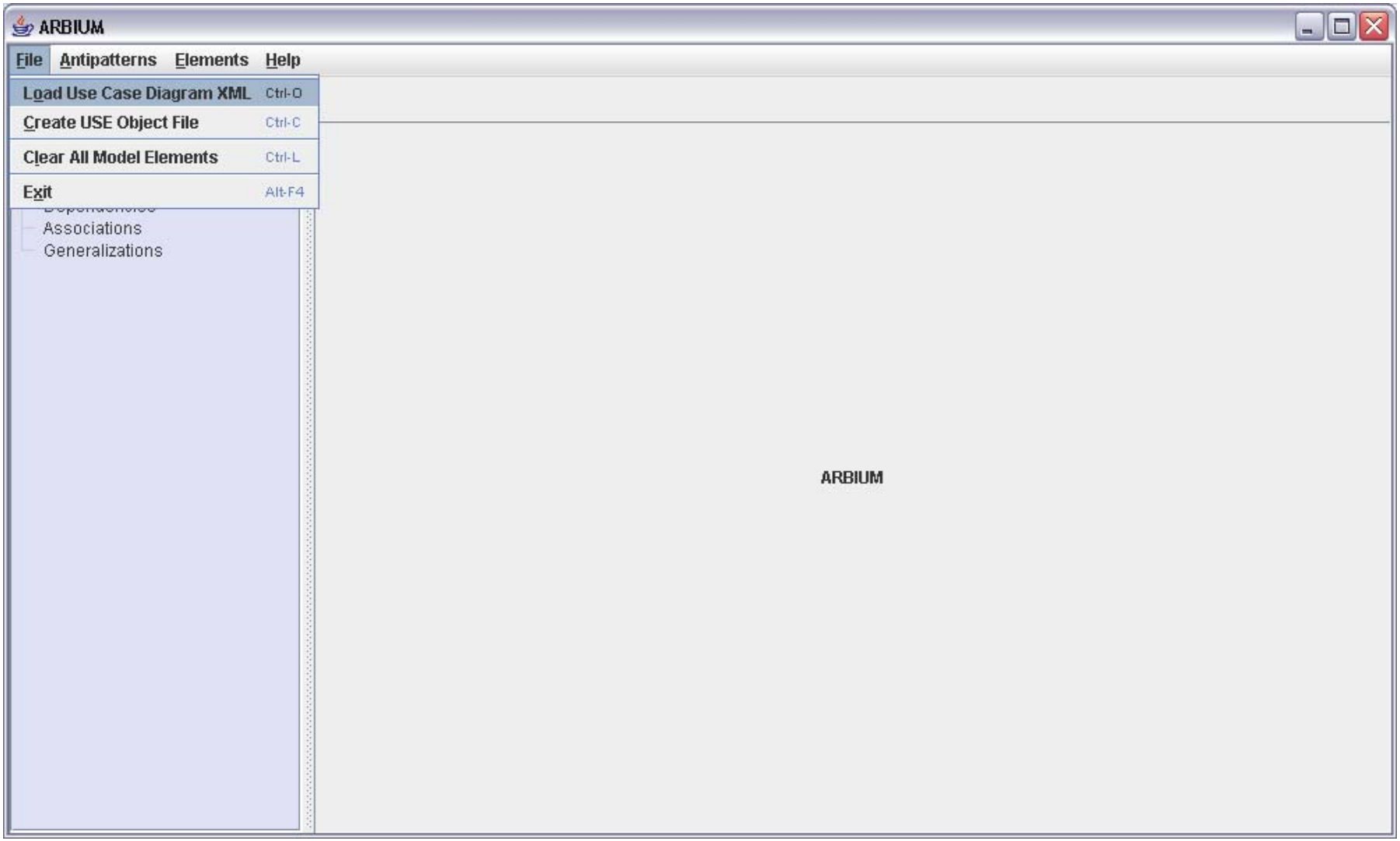
A summary of antipatterns(上)

Antipattern Name	Brief description
a1. UCs containing common + exceptional functionality	Use cases are used to contain different types of behavior.
	inv ExtendingMoreThanOneUseCase: `not (self.extended->size + self.extendedUC->size > 1)`
a2. Accessing a *generalized concrete* UC	An actor is set to access a parent UC that may contain incomplete functionality
	inv AccessingGeneralizedUseCaseByActor: `not ( (not (self.isAbstract)) and self.actorEnd->size > 0 and self.child->size >0)`
a3. Accessing an *extension* UC	An actor is set to access an extension UC for inappropriate reasons
	inv AccessingExtensionUseCaseByActor: `not( ( self.extended -> size > 0 or self.extendedUC->size>0) and self.actorEnd->size > 0 )`
a4. Using *extension* and/or *inclusion* UCs to implement an *abstract* UC	Using the *include* and *extend* relationships to implement an *abstract* UC. *Abstract* UCs are only properly implemented using the *generalization* relationship
	inv UsingIncludeAndExtendToImplementAbstractUC: `not((self.isAbstract) and (self.inclusion->size > 0 or self.extension->size > 0 or self.extensionUC->size>0))`
a5. Multiple actors associated with a UC	A UC that is associated with more than one actor whom all play the same role when the UC is being preformed
	inv MultipleActorsAssociatedWithUC: `not (self.actorEnd->size > 1)`

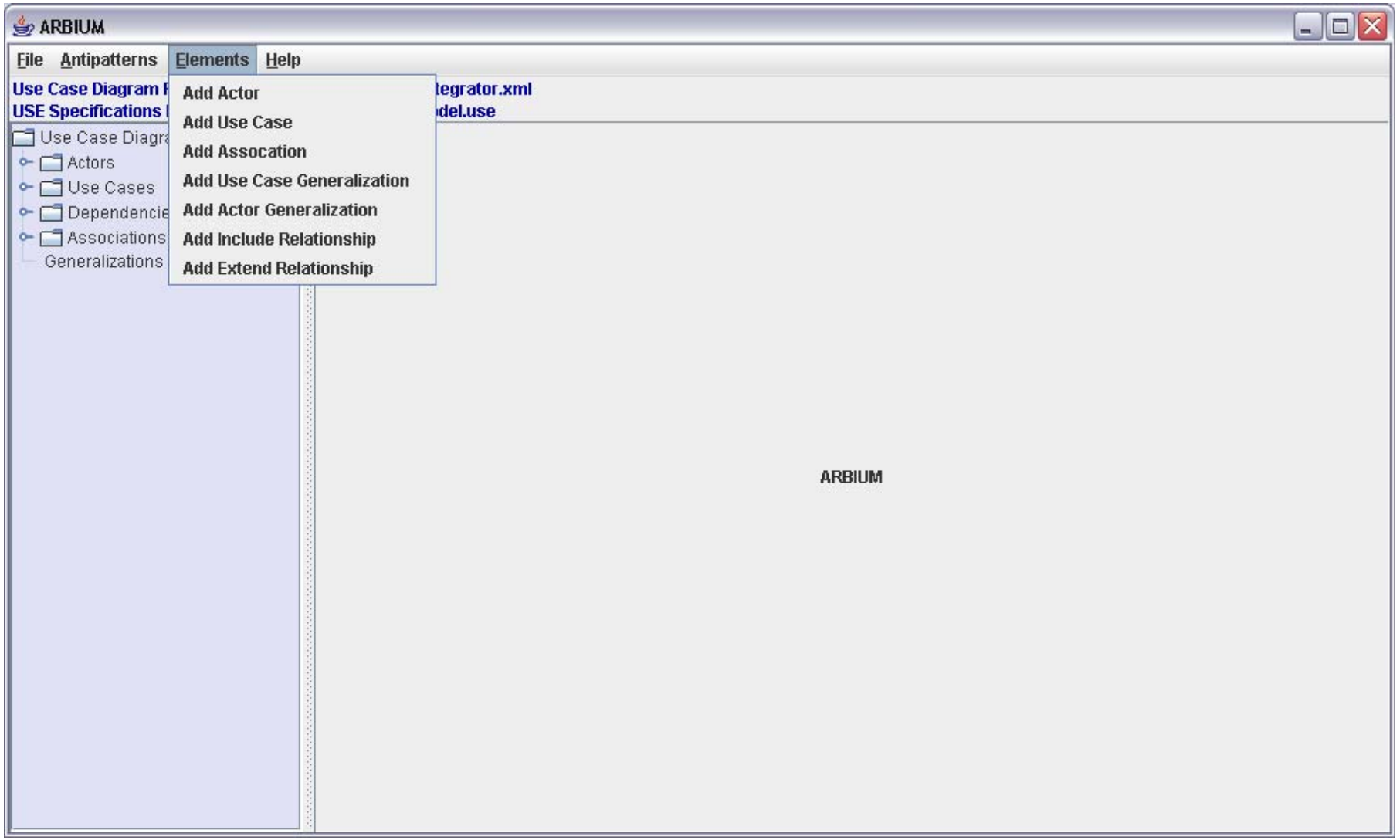
A summary of antipatterns(下)

a6. Multiple generalizations of a UC	A child UC is specializing more than one parent UC to contain common functionality between the parent UCs
	inv MultipleGeneralizationsOfOneUC: not (self.parent->size > 1)
a7. Duplicating relationships at the *generalized* and *specialized* UCs using the include and extend relationships instead.	Inappropriately using the include and extend relationships in addition to the generalization relationship in order to specify common behavior between parent and child UCs
	inv DupFuncAtChildAndParentUCUsingInclude: not (UseCase.allInstances->forAll (u1 , u2 | ((self <> u2) and (u1 <> u2) and (self <>u1))implies (self.inclusion->includes(u2) and u1.inclusion-> includes(u2)))) **inv DupFuncAtChildAndParentUCUsingExtend:** not (UseCase.allInstances->forAll (u1 , u2 | ((self <> u2) and (u1 <> u2) and (self <>u1))implies ((self.extended->includes(u2) or self.extendedUC-> includes(u2)) and(u1.extended-> includes(u2)or self.extendedUC->includes(u2)))
a8. Accessing an *abstract* UC that is not implemented	An *abstract* UC needs to be implemented in order for the specified behavior to be performed.
	inv AccessingUnimplementedAbstractUC: not ((self.actorEnd->size > 0) and self.child->size = 0) and self.isAbstract))

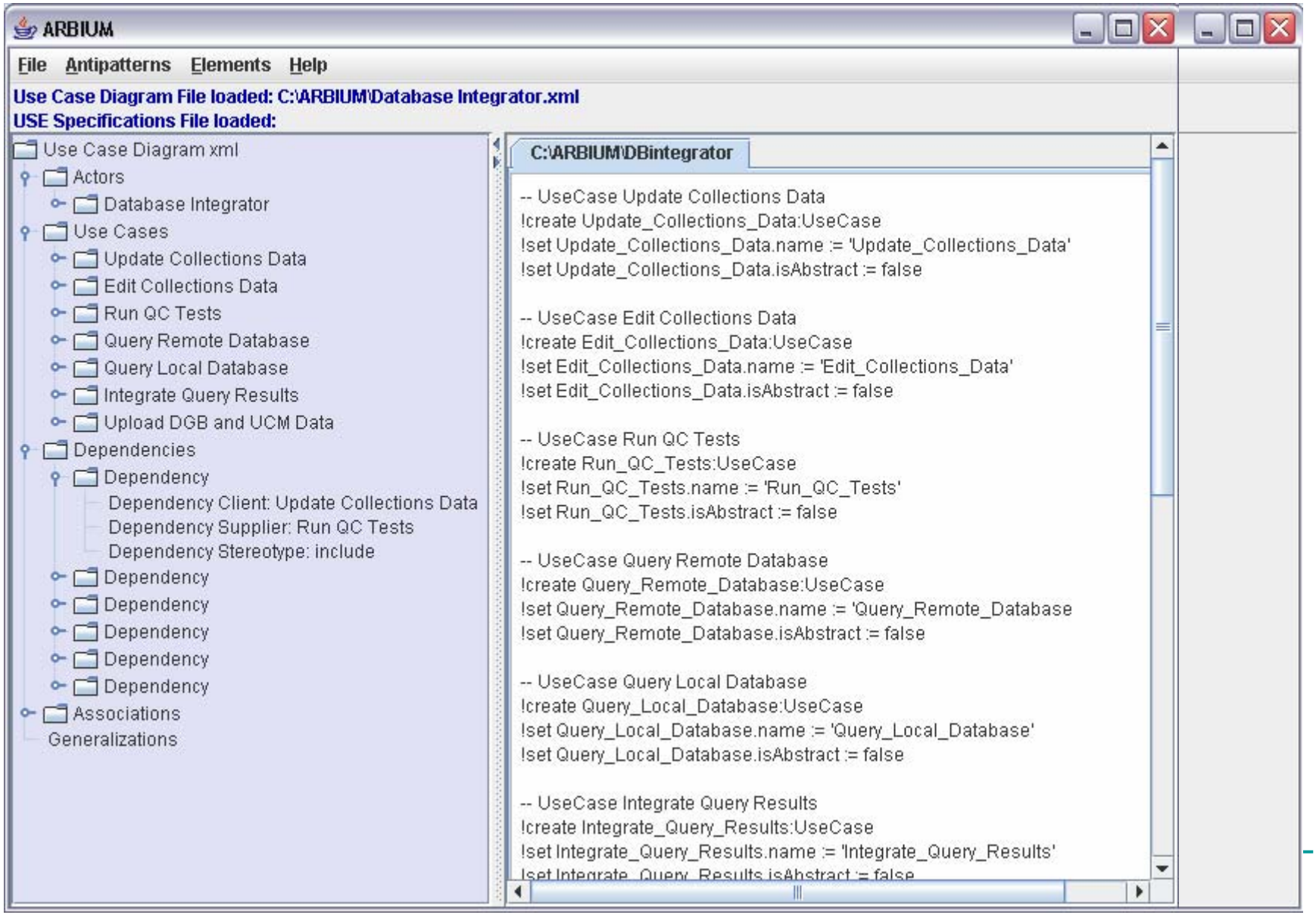
Load Use Case Diagram XML



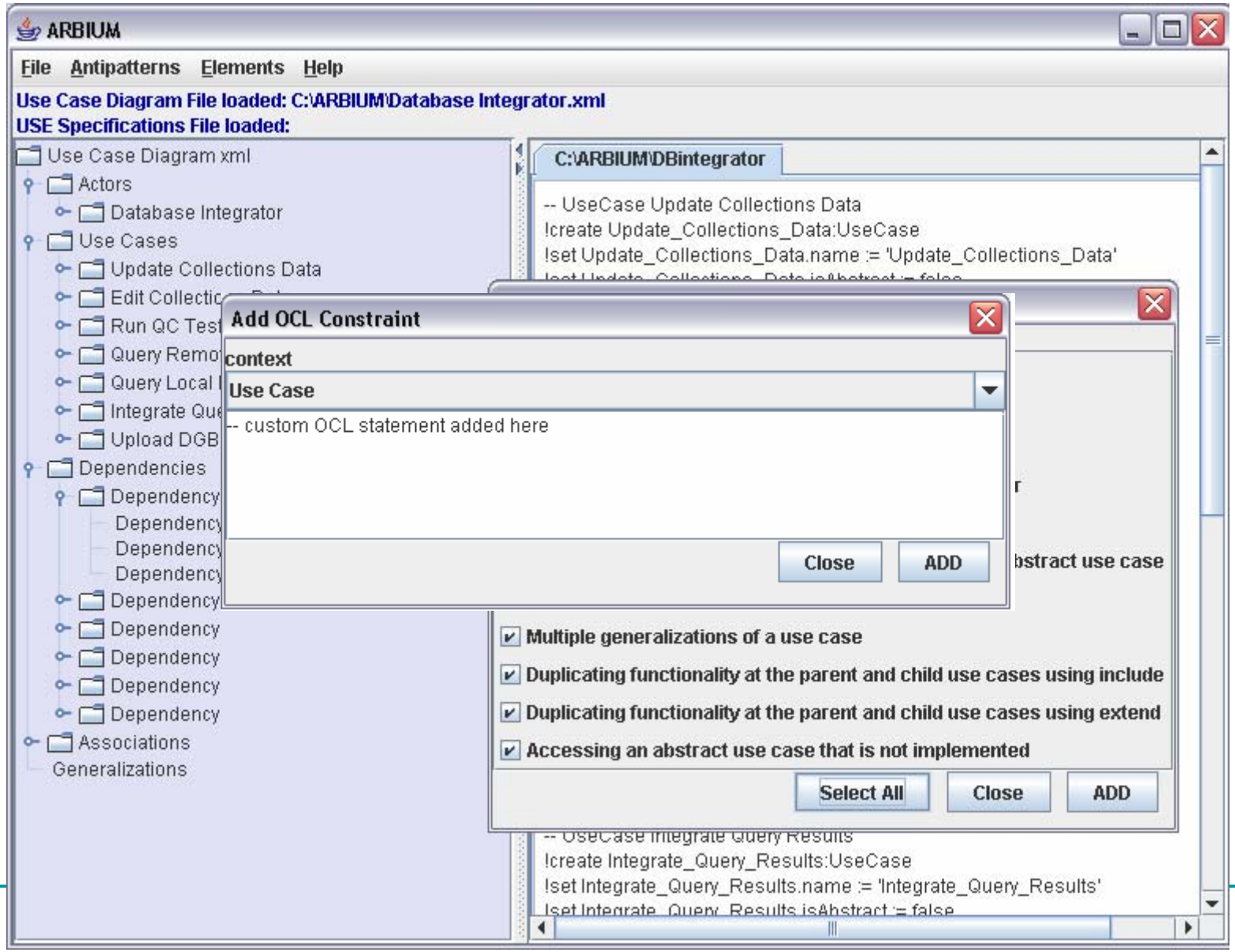
Enter user's UC diagram properties manually



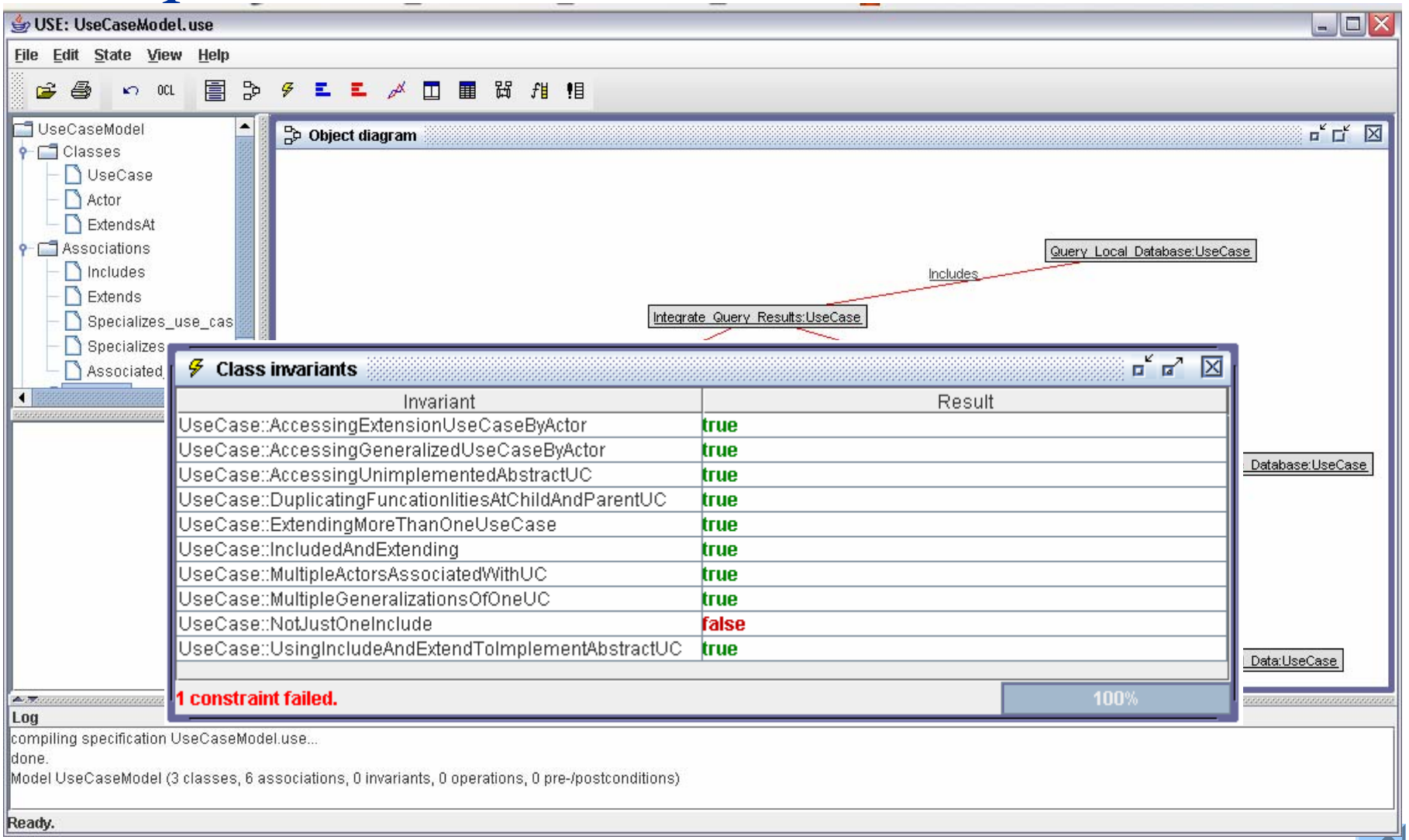
Create USE Object File



Add Antipattern

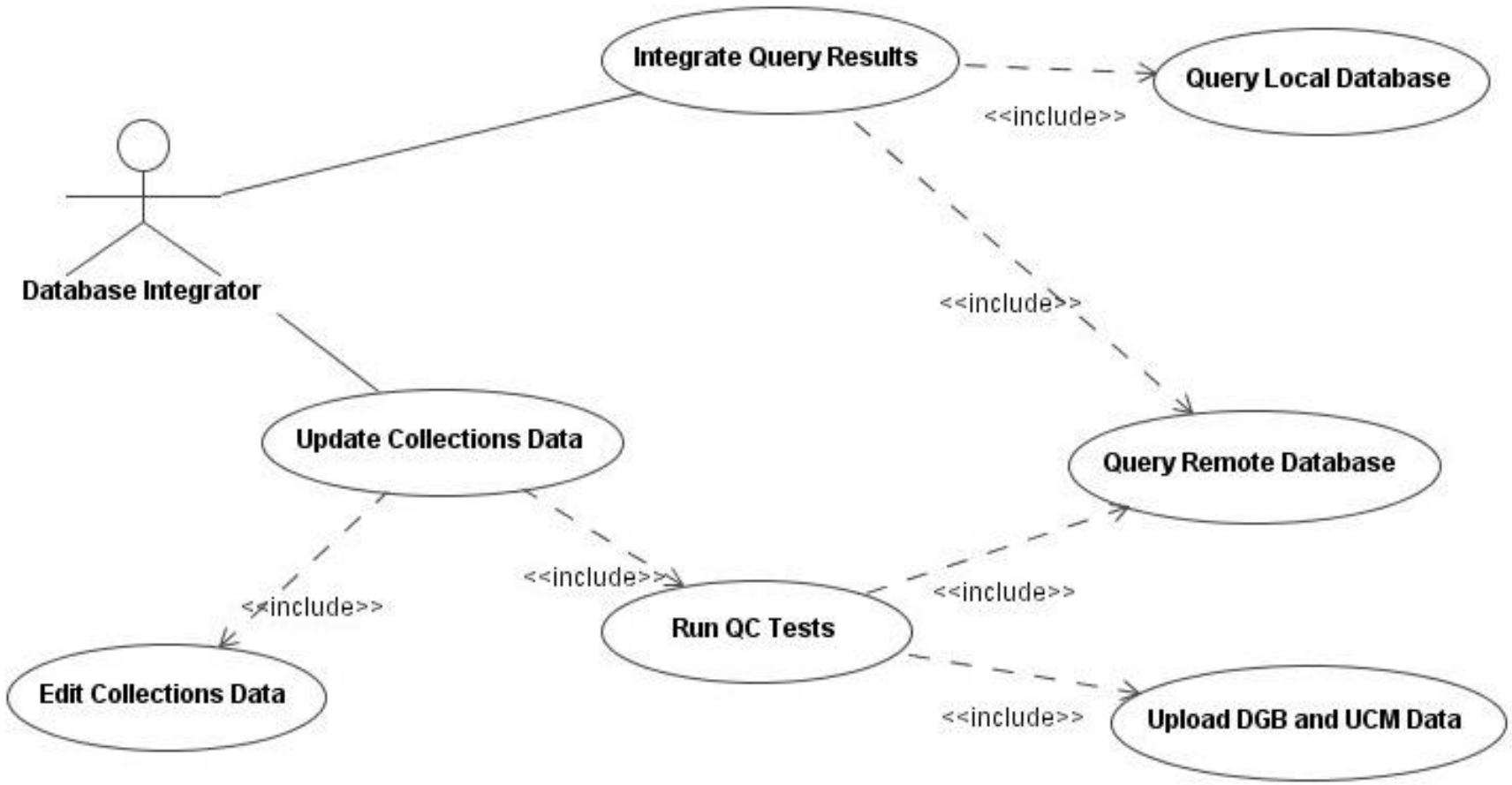


Result: Object diagram and wether match any Antipattern



Antipattern matches in each iteration

	UC Diagram	Antipattern Matched	Quality Improved – Potential Severity
First Iteration Matches	Database Access	Multiple actors associated with a UC	Correctness – very severe
	Database Queries	Functional decomposition: Using the *extend* relationship	Correctness – very severe
	Database Integrator	Functional decomposition: Using the *include* relationship	Analytical – non critical
		Functional decomposition: Using the *include* relationship	Analytical – non critical
	Database Edits	Functional decomposition: Using the *include* relationship	Analytical – non critical
		Accessing an *extending* UC	Correctness and Consistency – very severe
	Administrative Process	Multiple actors associated with a UC	Correctness – very severe
		Multiple actors associated with a UC	Correctness – very severe
		Multiple actors associated with a UC	Correctness – very severe
Second Iteration Matches	Merged UC Diagram	Accessing a *generalized concrete* UC	Correctness – very severe
		UCs containing common and exceptional functionality	False positive


Integrate Query Results
Query Local Database
<<include>>
Database Integrator
<<include>>
Update Collections Data
Query Remote Database
<<include>>
<<include>>
<<include>>
Run QC Tests
Edit Collections Data
<<include>>
Upload DGB and UCM Data