Agile Modeling

UML 2 Activity Diagramming Guidelines

In many ways UML Activity diagrams are the object-oriented equivalent of flow charts and data-flow diagrams (DFDs). They are used to explore the logic of:

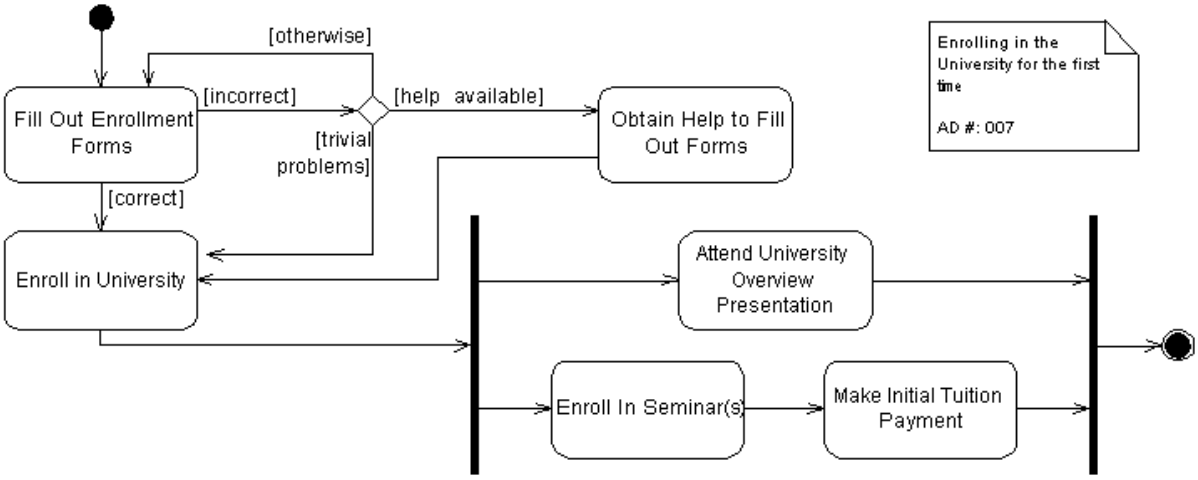
- A complex operation
- A complex business rule
- A single use case
- Several use cases
- A business process
- Software processes

Guidelines:

1. General Guidelines
2. Activities
3. Decision Points
4. Guards
5. Parallel Activities
6. Swimlanes
7. Action Objects

1. General Guidelines

Figure 1. Modeling a business process with a UML Activity Diagram.



1. Place The Start Point In The Top-Left Corner. A start point is modeled with a filled in circle, using the same notation that UML State Chart diagrams use. Every UML Activity Diagram should have a starting point, and placing it in the top-left corner reflects the way that people in Western cultures begin reading. Figure 1, which models the business process of enrolling in a university, takes this approach.
2. Always Include an Ending Point. An ending point is modeled with a filled in circle with a border around it, using the same notation that UML State Chart diagrams use. Figure 2 is interesting because it does not include an end point because it describes a continuous process - sometimes the guidelines don't apply.
3. Flowcharting Operations Implies the Need to Simplify. A good rule of thumb is that if an operation is so complex you need to develop a UML Activity diagram to understand it that you should consider refactoring it.

2. Activities

An activity, also known as an activity state, on a UML Activity diagram typically represents the invocation of an operation, a step in a business process, or an entire business process.

1. Question "Black Hole" Activities. A black hole activity is one that has transitions into it but none out, typically indicating that you have either missed one or more transitions.

2. Question "Miracle" Activities. A miracle activity is one that has transitions out of it but none into it, something that should be true only of start points.

3. Decision Points

A decision point is modeled as a diamond on a UML Activity diagram.

1. Decision Points Should Reflect the Previous Activity. In Figure 1 you see that there is no label on the decision point, unlike traditional flowcharts which would include text describing the actual decision being made, you need to imply that the decision concerns whether the person was enrolled in the university based on the activity that the decision point follows. The guards, depicted using the format *[description]*, on the transitions leaving the decision point also help to describe the decision point.
2. Avoid Superfluous Decision Points. The *Fill Out Enrollment Forms* activity in Figure 1

4. Guards

A guard is a condition that must be true in order to traverse a transition.

1. Each Transition Leaving a Decision Point Must Have a Guard
2. Guards Should Not Overlap. For example guards such as x <0, x = 0, and x > 0 are consistent whereas guard such as x <= 0 and x >= 0 are not consistent because they overlap - it isn't clear what should happen when x is 0.
3. Guards on Decision Points Must Form a Complete Set. For example, guards such as x < 0 and x >0 are not complete because it isn't clear what happens when x is 0.
4. Exit Transition Guards and Activity Invariants Must Form a Complete Set. An activity invariant is a condition that is always true when your system is processing an activity.
5. Apply a [Otherwise] Guard for "Fall Through" Logic.
6. Guards Are Optional. It is very common for a transition to not include a guard, even when an activity includes several exit transitions. Follow Agile Modeling (AM)'s principle of Depict Models Simply and only indicate a guard on a transition if it adds value.

5. Parallel Activities

It is possible to show that activities can occur in parallel, as you see in Figure 1 depicted using two parallel bars. The first bar is called a fork, it has one transition entering it and two or more transitions leaving it. The second bar is a join, with two or more transitions entering it and only one leaving it.

1. A Fork Should Have a Corresponding Join. In general, for every start (fork) there is an end (join). In UML 2 it is not required to have a join, but it usually makes sense.
2. Forks Have One Entry Transition.
3. Joins Have One Exit Transition
4. Avoid Superfluous Forks. Figure 2 depicts a simplified description of the software process of enterprise architectural modeling, a part of the Enterprise Unified Process (EUP). There is significant opportunity for parallelism in this process, in fact all of these activities could happen in parallel, but forks were not introduced because they would only have cluttered the diagram.

6. Swimlane Guidelines

A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread. Figure 2 includes three swimlanes, one for each actor.

Figure 2. A UML activity diagram for the enterprise architectural modeling (simplified).

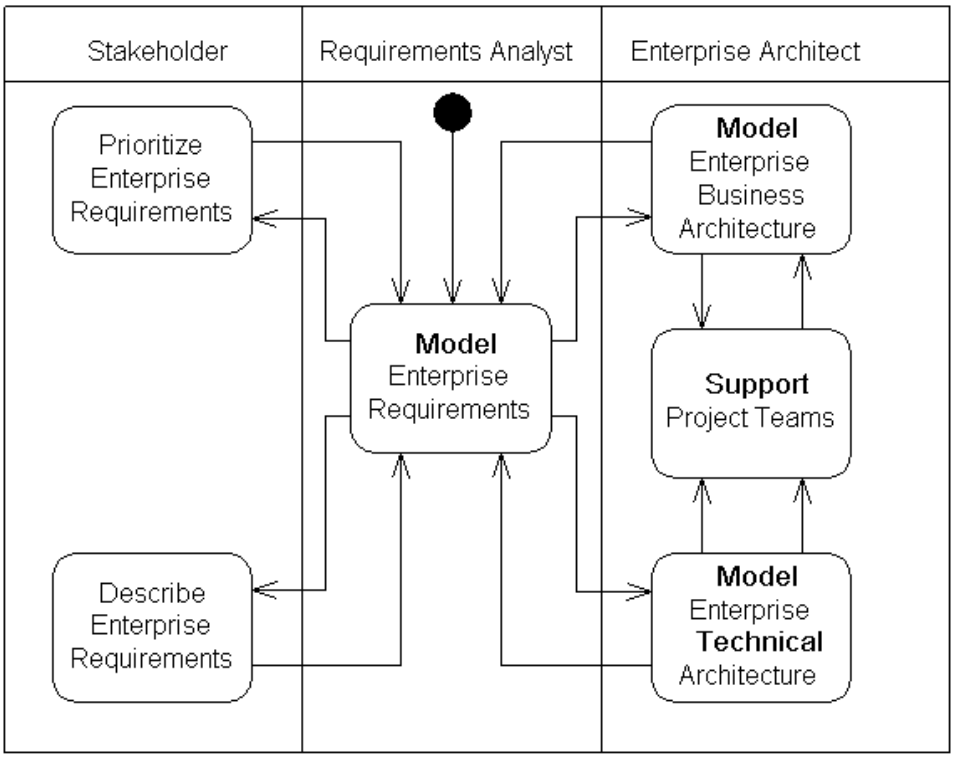
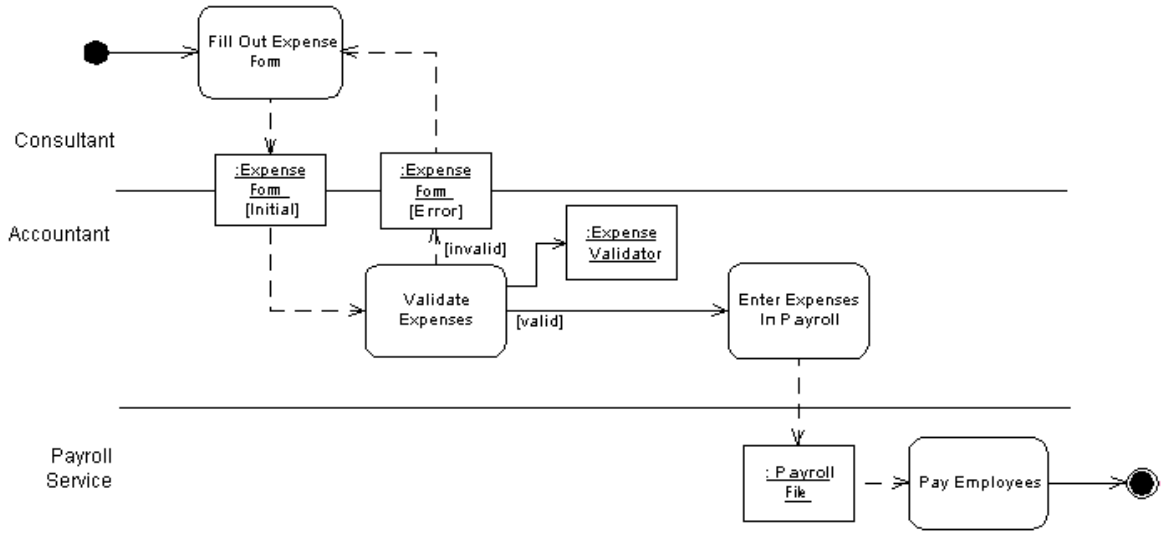


Figure 3. Submitting expenses.



1. Order Swimlanes in a Logical Manner.
2. Apply Swim Lanes To Linear Processes. A good rule of thumb is that swimlanes are best applied to linear processes, unlike the one depicted in Figure 2.
3. Have Less Than Five Swimlanes.
4. Consider Swimareas For Complex Diagrams.
5. Swimareas Suggest The Need to Reorganize Into Smaller Activity Diagrams.
6. Consider Horizontal Swimlanes for Business Processes. In Figure 3you see that the swimlanes are drawn horizontally, going against common convention of drawing them vertically.

7 Action-Object Guidelines

Activities act on objects, In the strict object-oriented sense of the term an action object is a system object, a software construct. In the looser, and much more useful for business application modeling, sense of the term an action object is any sort of item. For example in Figure 3the *ExpenseForm* action object is likely a paper form.

1. Place Shared Action Objects on Swimlane Separators
2. When An Object Appears Several Time Apply State Names
3. State Names Should Reflect the Lifecycle Stage of an Action Object

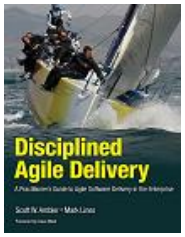
4. Show Only Critical Inputs and Outputs
5. Depict Action Objects As Smaller Than Activities

Let Us Help

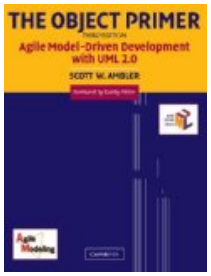
We actively work with clients around the world to improve their information technology (IT) practices, typically in the role of mentor/coach, team lead, or trainer. A full description of what we do, and how to contact us, can be found at Scott Ambler + Associates.

Recommended Reading

The Elements of UML 2.0 Style describes a collection of standards, conventions, and guidelines for creating effective UML diagrams . They are based on sound, proven software engineering principles that lead to diagrams that are easier to understand and work with. These conventions exist as a collection of simple, concise guidelines that if applied consistently, represent an important first step in increasing your productivity as a modeler. This book is oriented towards intermediate to advanced UML modelers, although there are numerous examples throughout the book it would not be a good way to learn the UML (instead, consider The Object Primer). The book is a brief 188 pages long and is conveniently pocket-sized so it's easy to carry around.



This book, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise describes the Disciplined Agile Delivery (DAD) process decision framework. The DAD framework is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value delivery lifecycle, is goal-driven, is enterprise aware, and provides the foundation for scaling agile. This book is particularly important for anyone who wants to understand how agile works from end-to-end within an enterprise setting. Data professionals will find it interesting because it shows how agile modeling and agile database techniques fit into the overall solution delivery process. Enterprise professionals will find it interesting beause it explicitly promotes the idea that disciplined agile teams should be enterprise aware and therefore work closely with enterprise teams. Existing agile developers will find it interesting because it shows how to extend Scrum-based and Kanban-based strategies to provide a coherent, end-to-end streamlined delivery process.



The Object Primer 3rd Edition: Agile Model Driven Development with UML 2 is an important reference book for agile modelers, describing how to develop 35 types of agile models including all 13 UML 2 diagrams. Furthermore, this book describes the fundamental programming and testing techniques for successful agile solution delivery. The book also shows how to move from your agile models to source code, how to succeed at implementation techniques such as refactoring and test-driven development(TDD). The Object Primer also includes a chapter overviewing the critical database development techniques (database refactoring, object/relational mapping, legacy analysis, and database access coding) from my award-winning Agile Database Techniquesbook.