<table>
<thead>
<tr>
<th>S</th>
<th>W</th>
<th>Name</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ese-2013-team1</td>
<td>N/A</td>
<td>7 hr 17 min - #7</td>
<td>3 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team2</td>
<td>N/A</td>
<td>18 hr - #9</td>
<td>4 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team3</td>
<td>14 hr - #15</td>
<td>2 days 16 hr - #7</td>
<td>2 min 14 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team4</td>
<td>1 day 18 hr - #13</td>
<td>18 hr - #14</td>
<td>2 min 3 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team5</td>
<td>N/A</td>
<td>5 hr 3 min - #7</td>
<td>4 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team6</td>
<td>20 min - #14</td>
<td>3 days 0 hr - #9</td>
<td>5 min 0 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team7</td>
<td>N/A</td>
<td>23 hr - #9</td>
<td>41 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team8</td>
<td>13 hr - #15</td>
<td>2 days 13 hr - #13</td>
<td>2 min 16 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ese-2013-team9</td>
<td>N/A</td>
<td>21 hr - #10</td>
<td>2 min 11 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PomodoroBox</td>
<td>6 hr 45 min - #36</td>
<td>N/A</td>
<td>2 min 18 sec</td>
</tr>
</tbody>
</table>
Introduction to Software Engineering

8. UML

Mircea F. Lungu
Galactic Modeling Language

The GML (GalacticModelingLanguage) is a modeling language with three elements:

- The Box
- The Line
- The Label

GML diagrams mean what they mean when they are scribbled. The preferred GML CaseTool is a hotel pen and a beer-stained cocktail napkin.

Proposed by Kent Beck, not entirely a joke. E.g. slashdot story *UML Really Necessary?* : 'A while ago I saw Kent Beck talk at the Java user's group meeting here in Seattle. Someone asked him about UML. He made a derisive noise and sneered that he had come up with a better version called GML, Galactic Modeling Language. He said (and I am paraphrasing here) that GML had three components "Boxes, Arrows and Arrows Pointing to Boxes".'

*I prostrate myself in honor of someone who has finally cut through all the crap. -- Philip*

*Hear, hear.*

See also AdvancedFactoring.
Who knows UML?
Who can name more than 3 types of diagrams?

1. ...
2.
UML Diagram Types

Structural Diagrams
- Composite Structure Diagram
- Deployment Diagram
  - Package Diagram
  - Profile Diagram
  - Class Diagrams
  - Object Diagram
  - Component Diagrams

Behavioral Diagrams
- State Machine Diagram
- Communication Diagram
  - Use Case Diagram
  - Activity Diagram
  - Sequence Diagram
  - Timing Diagram
  - Interaction Overview Diagram
UML Class Diagram Quiz!
CD-Q1. What design pattern is this?
CodeBubbles also allows an overview of a system. And it’s executable.
What is UML?
**UML**

**What?**
- uniform notation: Booch + OMT + Use Cases (+ state charts)

**Why?**
- Software projects are carried out in *team*
- Team members need to *communicate*
  — ... sometimes even with the end users
- “One picture conveys a thousand words”
  — the question is only *which words*
  — Need for *different views* on the same software artifact
thinking tool
communication tool (with stakeholders)
collaboration tool (dialog with designers)
adaptation (i.e., using a homegrown variant of the "real" notation)
selective traction (i.e., using it just as long as is useful, then moving on)
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State
Further Discussion
Class Diagrams

Figure 3-1. Class diagram
Visibility and Scope of Features

**Stereotype**
(what “kind” of class is it?)

**User-defined properties**
(e.g., readonly, owner = “Pingu”)

Underlined attributes have **class scope**

+ = “public”
# = “protected”
- = “private”

An ellipsis signals that further entries are not shown

```
Window
{ abstract }

+size: Area = (100, 100)
#visibility: Boolean = false
+default-size: Rectangle
#maximum-size: Rectangle
-xptr: XWindow*

+display ()
+hide ()
+create ()
-attachXWindow (xwin: Xwindow*)
...```

Italic attributes are **abstract**
<table>
<thead>
<tr>
<th>UML Lines and Arrows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constraint</strong></td>
</tr>
<tr>
<td>(usually annotated)</td>
</tr>
<tr>
<td><strong>Association</strong></td>
</tr>
<tr>
<td>e.g., «uses»</td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
</tr>
<tr>
<td>e.g., «requires»,</td>
</tr>
<tr>
<td>«imports» ...</td>
</tr>
<tr>
<td><strong>Navigable</strong></td>
</tr>
<tr>
<td>association</td>
</tr>
<tr>
<td>e.g., part-of</td>
</tr>
<tr>
<td><strong>Realization</strong></td>
</tr>
<tr>
<td>e.g., class/template,</td>
</tr>
<tr>
<td>class/interface</td>
</tr>
<tr>
<td><strong>“Generalization”</strong></td>
</tr>
<tr>
<td>i.e., specialization (!)</td>
</tr>
<tr>
<td>e.g., class/superclass,</td>
</tr>
<tr>
<td>concrete/abstract class</td>
</tr>
<tr>
<td><strong>Aggregation</strong></td>
</tr>
<tr>
<td>i.e., “consists of”</td>
</tr>
<tr>
<td><strong>“Composition”</strong></td>
</tr>
<tr>
<td>i.e., containment</td>
</tr>
</tbody>
</table>
Parameterized (aka “template” or “generic”) classes are depicted with their parameters shown in a **dashed box**.

**Figure 13-180.** Template notation with use of parameter as a reference
Interfaces

Interfaces, equivalent to abstract classes with no attributes, are represented as classes with the stereotype «interface» or, alternatively, with the “Lollipop-Notation”:

![Diagram of interface realization](image)

**Figure B-5.** Realization of an interface
Generalization

A **subclass** specializes its superclass:

Figure 4-7. Generalization notation
The different faces of inheritance

Is-a

Polymorphism

Reuse
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State
Further Discussion
Objects
**Associations** represent *structural relationships* between objects

—usually *binary* (but may be ternary etc.)

—optional *name* and *direction*

—(unique) *role names* and *multiplicities* at end-points

Figure 4-2. Association notation
## Multiplicity

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>Zero or one entity</td>
</tr>
<tr>
<td>1</td>
<td>Exactly one entity</td>
</tr>
<tr>
<td>*</td>
<td>Any number of entities</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more entities</td>
</tr>
<tr>
<td>1..n</td>
<td>One to n entities</td>
</tr>
</tbody>
</table>

*And so on …*
Aggregation and Composition

**Aggregation** is denoted by a *diamond* and indicates a *part-whole dependency*:

A *hollow diamond* indicates a *reference*; a *solid diamond* an *implementation* (i.e., ownership).

**Aggregation**: parts may be shared.

**Composition**: one part belongs to one whole.

*Figure 13-29. Various adornments on association ends*
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State
Further Discussion
Sequence Diagram, e.g.
The Elements of a Sequence Diagram
Activations

Figure 8-2. Sequence diagram with activations
Asynchrony and Constraints

constraints
\{b - a < 1 \text{ sec.}\}

\{c - b < 10 \text{ sec.}\}

comment
The call is routed through the network.

\{d' - d < 5 \text{ sec.}\}

At this point, the parties can talk.

Figure 13-161. Sequence diagram with asynchronous control
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State
Further Discussion
Communication Diagrams

Figure 8-3. Collaboration diagram
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State
Further Discussion
Figure 7-1. Activity diagram
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State
Further Discussion
**Statechart Diagrams**

**Figure 3-5. Statechart diagram**
A Statechart Diagram describes the temporal evolution of an object of a given class in response to interactions with other objects inside or outside the system.

An event is a one-way (asynchronous) communication from one object to another:
- *atomic* (non-interruptible)
- includes events from *hardware* and real-world objects e.g., message receipt, input event, elapsed time, ...
- notation: `eventName(parameter: type, ...)`
- may cause object to make a transition between states
A **state** is a period of time during which an object is **waiting** for an event to occur:

— depicted as *rounded box* with (up to) three sections:
  - name — *optional*
  - state variables — *name: type = value (valid only for that state)*
  - triggered operations — *internal transitions and ongoing operations*

— may be *nested*
The *entry event* occurs whenever a transition is made into this state, and the *exit operation* is triggered when a transition is made out of this state.

The *help* and *character* events cause internal transitions with no change of state, so the entry and exit operations are not performed.
A transition is an *response to an external event* received by an object in a *given state*

— May *invoke* an operation, and cause the object to change state
— May *send* an event to an external object
— Transition syntax (each part is optional):
  
  - `event(arguments) [condition]`
  - `/ ^target.sendEvent operation(arguments)`

— *External transitions* label arcs between states
— *Internal transitions* are part of the triggered operations of a state
Operations and Activities

An operation is an \textit{atomic action} invoked by a transition
—\textit{Entry and exit operations} can be associated with states

An activity is an \textit{ongoing operation} that takes place while object is in a given state
—Modelled as “internal transitions” labelled with the pseudo-event \textit{do}
Nested Statecharts

Figure 13-169. State diagram
Composite States

Composite states may be depicted either as high-level or low-level views.

“Stubbed transitions” indicate the presence of internal states:

*Initial and terminal substates* are shown as black spots and “bulls-eyes”

---

*Figure 13-172. Stubbed transition*
Sending Events between Objects

Each signal is directed to a specific object. This signal turns the VCR off or on, depending on its current state.

This signal turns the TV off or on, depending on its current state.

Figure 13-160. Sending signals between objects
Concurrent Substates

![State machine with concurrent composite state](image)

**Figure 6-6.** State machine with concurrent composite state
Branching and Merging

**Entering concurrent states:**
Entering a state with concurrent substates means that *each of the substates is entered concurrently* (one logical thread per substate).

**Leaving concurrent states:**
A *labelled transition* out of any of the substates *terminates all of the substates.*
An *unlabelled transition* out of the overall state *waits* for all substates to terminate.
Completing a Course

- **Doing Exercises**: Complete exercises.
- **Completed Exercises**: Adjust final grade.
- **Incomplete**: Insufficient exercises.
- **Passed Course**: Take repeat exam.
- **Failed Course**: Request «Nachprüfung».
- **Registered for Exam**: Take scheduled exam.
- **Completed**: Registered for exam.
- **Register (ePub)**: Notify Dozent.
- **Failed 1st Exam**: Request «Wiederholung».
- **Failed**: Don't show up.
- **Passed**: Show up.

**Timeout**
- Up to 14 days before exam
- Within 3 weeks email Dozent

**Is it correct?**
Roadmap

UML Overview
Structural Diagrams
  Classes, attributes and operations
  Objects, Associations
Behavioral Diagrams
  Sequence
  Communication
  Activity
  State

Further Discussion
Constraints are restrictions on values attached to classes or associations.

Figure 4-12. Constraints


> *UML in Practice*, Marian Petre, ICSE 2013

What you should know!

> Why do scenarios depict objects but not classes?
> How can timing constraints be expressed in scenarios?
> How do you use nested state diagrams to model object behavior?
> What is the difference between “external” and “internal” transitions?
> How can you model interaction between state diagrams for several classes?
> How do you represent classes, objects and associations?
> How do you specify the visibility of attributes and operations to clients?
> Why is inheritance useful in analysis? In design?
Can you answer the following questions?

> Can a sequence diagram always be translated to an communication diagram?
> Why are arrows depicted with the message labels rather than with links?
> How is aggregation different from any other kind of association?
> How are associations realized in an implementation language?
You are free:
▪ to copy, distribute, display, and perform the work
▪ to make derivative works
▪ to make commercial use of the work

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

▪ For any reuse or distribution, you must make clear to others the license terms of this work.
▪ Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.