CSC301: Introduction to Software Engineering

Lecture 7

Wael Aboulsaadat

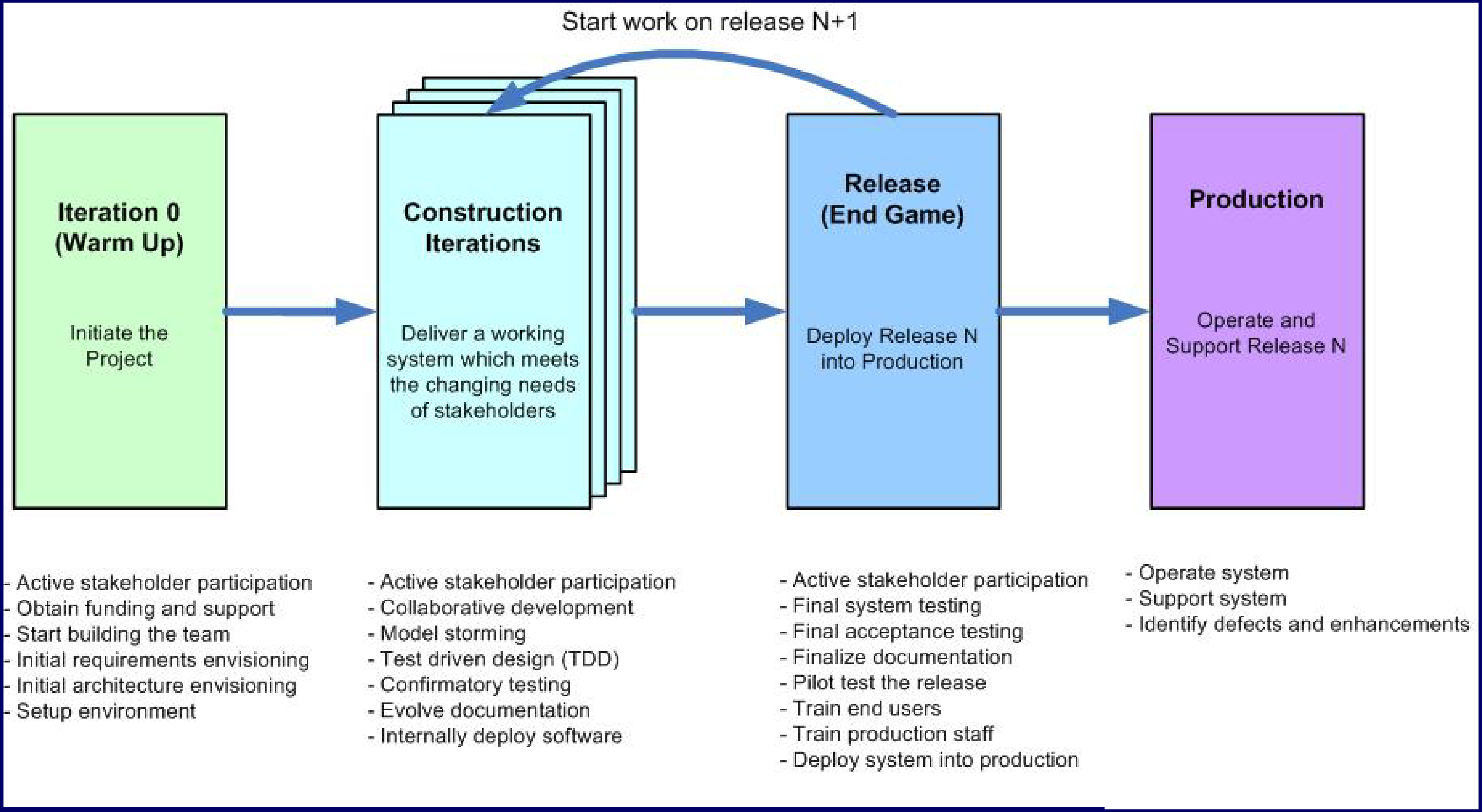
Agile SDLC

History: How did "agile" arise

- "Agile" techniques were in use since the beginning.
- Agile (*mobility-based*) techniques did not show competitive advantage in the 1970s / 1980s, but did during the 1990s and do now.
- 1994: trials of *semi*-formal agile methodologies

RAD	DSDM
XP	Crystal
Scrum	Adaptive

Agile SDLC


Start work on release N+1
Iteration 0 (Warm Up)
Initiate the Project
Construction Iterations
Deliver a working system which meets the changing needs of stakeholders
Release (End Game)
Deploy Release N into Production
Production
Operate and Support Release N

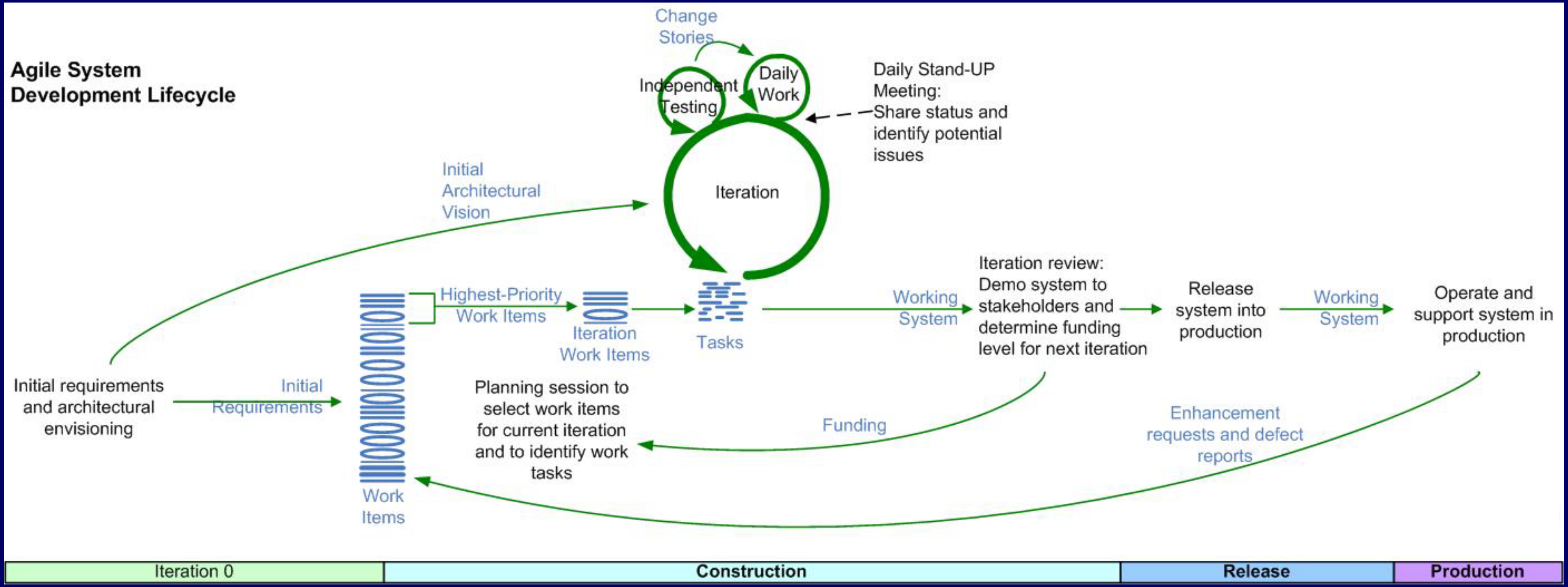
- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements envisioning
- Initial architecture envisioning
- Setup environment

- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Confirmatory testing
- Evolve documentation
- Internally deploy software

- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production

- Operate system
- Support system
- Identify defects and enhancements

Agile SDLC


Agile System Development Lifecycle
Change Stories
Independent Testing
Daily Work
Daily Stand-UP Meeting: Share status and identify potential issues
Iteration
Initial Architectural Vision
Highest-Priority Work Items
Iteration Work Items
Tasks
Working System
Iteration review: Demo system to stakeholders and determine funding level for next iteration
Release system into production
Working System
Operate and support system in production
Initial requirements and architectural envisioning
Initial Requirements
Work Items
Planning session to select work items for current iteration and to identify work tasks
Funding
Enhancement requests and defect reports
Iteration 0
Construction
Release
Production

Agile SDLC: core concepts

1) Pair Programming

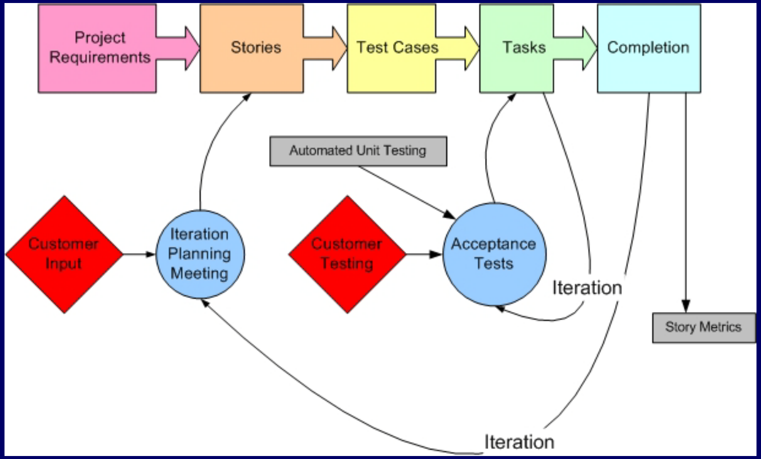
– people work best in pairs so that they complement each other's strengths and weaknesses

– Benefits: Economics, Satisfaction, Design quality, Continuous Reviews, Problem solving, Learning, Team Building and Communication, Staff and Project Management

Agile SDLC: core concepts

2) Stories and Test/Requirement-driven development

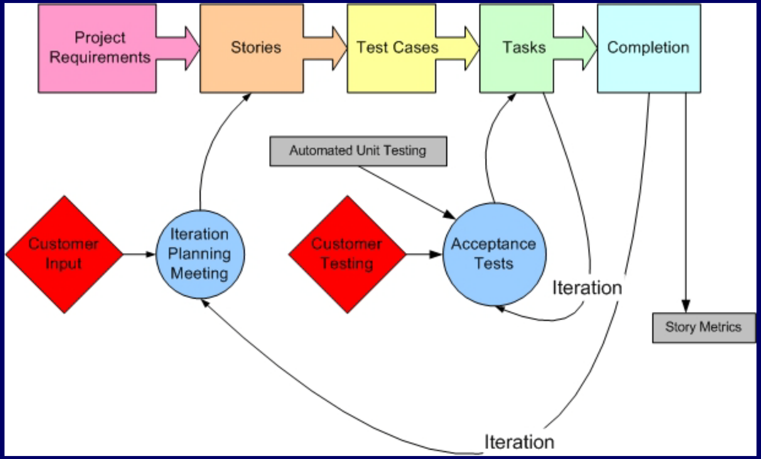
- The project is done in iterations (1 – 3 weeks). At the end of an iteration, the next iteration is planned out in an iteration planning meeting. The customer (with the developers' guidance, of course) decides what stories are to be implemented in the next iteration, based on what is most important to them, and what is most practical to implement next.



Agile SDLC: core concepts

2) Stories and Test/Requirement-driven development

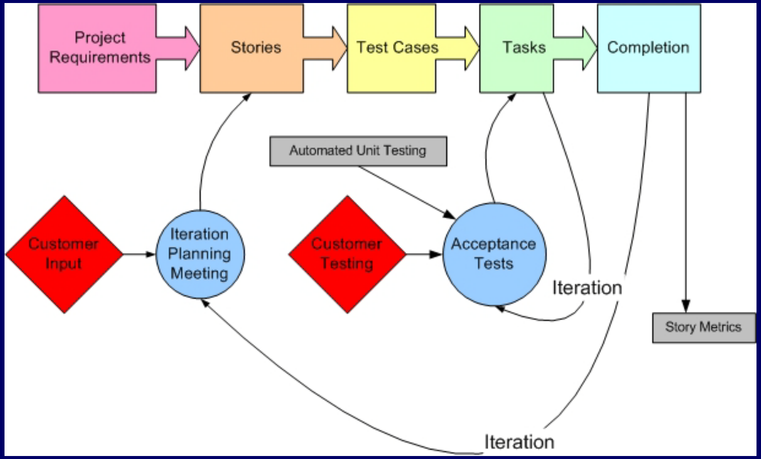
- A project's requirements are laid out as individual stories. These stories have test cases. A test case includes the test and the desired results which, if met, indicate that that story is complete.
- The stories are broken down into tasks, which are the steps needed to implement a given user story. Developers then sign up to do those tasks.



Agile SDLC: core concepts

2) Stories and Test/Requirement-driven development

- On the completion of an iteration, acceptance tests are done to ensure that the test cases defined in that story have been met. The acceptance tests include both automated unit testing, and customer testing. This makes sure that all requirements are reached because the development is based around those requirements and there are definite indicators on whether a given requirement is met. It also breaks down the project into reachable goals rather than having programmers work away forever on an ever-expanding megalith.



Agile SDLC: core concepts

3) Simplicity

- Go the simplest way that meets the requirements - complexity may be clever, but it costs more in the end (maintenance, etc). No needless complications.

Agile SDLC: core concepts

4) Design as you go!

- This development method enables the design to evolve as you implement the stories and get an in-depth look at what is needed to fill the requirements.
- This will, of course, require a lot of refactoring of code… but it also will lead to the flexibility to add new features as they come.
- Needless to say, JIT Designing is one of the most controversial concepts in agile SDLC...

Agile SDLC: core concepts

4) Design as you go!

- This development method enables the design to evolve as you implement the stories and get an in-depth look at what is needed to fill the requirements.
- This will, of course, require a lot of refactoring of code… but it also will lead to the flexibility to add new features as they come.
- Warning: needless to say, JIT Designing is one of the most controversial concepts in agile SDLC...
  - It can become counter-productive to design as you go because certain things need to be planned out ahead of any implementation. With no concrete design, things can become chaos.
  - However, there are many things which, if planned ahead, will only be re-designed later on.

Agile SDLC: core concepts

5) Close collaboration/feedback

- A daily stand-up meeting of all members of the group.
  - At some point in each day, a stand-up meeting is held where everyone stands up in a circle and talks about the project and its progress.
- Swapping developers between tasks, so everyone gets to know all the different parts of the system being developed.

Agile SDLC: core concepts

6) Continuous Integration

- Eliminates the headaches of working with outdated code and trying to integrate fragmented or diverging code into the system.

Agile SDLC: core concepts

7) Sustainable Pace

- maintain a moderate pace that can be sustained throughout the entire length of the project.
    - No caffeine night-dose, no free chocolate

Agile SDLC: core concepts

8) Onsite customer

- Have the customer be as available and as close as possible, preferably on-site.
  - Warning: It is unlikely that it will be possible for a competent customer representative to be available at all times… ☹

Agile SDLC: core concepts

9) Collective Code ownership

- Anyone in the developer team should be able to change any piece of code.
  - Warning: If the person who changed the code does not talk to others both before and after the changes, problems can arise

Agile SDLC: core concepts

10) Refactor Mercilessly

- When you see something that could be better and simpler, change it!
  - Warning:
    - If you're constantly changing things, it can be hard for everyone to keep up with the changes
    - Regression testing is a must with constant refactoring. The reliance on testing becomes a weak point

Refactoring

What is Refactoring?

- The process of *changing a software system* in such a way that it *does not alter the external behaviour* of the code, yet *improves its internal structure*.

  – Fowler, et al., Refactoring, 1999.

Typical Refactorings

Class Refactorings	Method Refactorings	Attribute Refactorings
add (sub)class to hierarchy	add method to class	add variable to class
rename class	rename method	rename variable
remove class	remove method	remove variable
	push method down	push variable down
	push method up	pull variable up
	add parameter to method	create accessors
	move method to component	abstract variable
	extract code in new method	

How to do refactoring

- Need to know about design patterns
- Need to know about anti-patterns
- Need to know about refactoring patterns

- You can start by checking for bad-smell in code!

There are at least 192 design patterns, 72 analysis patterns,
42 anti-patterns, and 70 refactoring patterns

Bad smell in code: within classes

- Lots of comments
- Long method
- Large Class
- Long Parameter List
- Duplicate Code
- Conditional complexity
- Checking on type
- Inconsistent/uncommuincative names
- Dead code

Bad smell in code: in-between classes

- Primitive Obsession
- Data Class
- Data Clumps
- Refused Bequest
- Inappropriate Intimacy
- Indecent Exposure
- Shotgun Surgery
- And many more…

Refactoring patterns: classification

- Composing methods
- Moving features between objects
- Organizing Data
- Simplifying conditional expressions
- Making method calls simpler
- Dealing with generalization
- Big effort

Refactoring patterns

→Read all patterns here:
http://sourcemaking.com/refactoring

What is an AntiPattern?

- Describing AntiPatterns
  - Symptoms and Consequences
  - Typical Causes
  - Known Exceptions
  - Refactored Solutions
  - Variations
- Anti-patterns classification
  - Architecture anti-patterns
  - Development anti-patterns
  - Management anti-patterns

http://c2.com/cgi/wiki?AntiPatternsCatalog

Helpers for agile methodology

- Smart IDEs
- Hungarian notation
- Code Annotation

Helpers for agile methodology: Smart IDEs

- E.g. IntelliJ IDE, eclipse, netbeans

Helpers for agile methodology: Hungarian notation

- A useful naming convention for attributes and methods
- http://en.wikipedia.org/wiki/Hungarian_notation

Helpers for agile methodology: Code Annotation

- Documentation

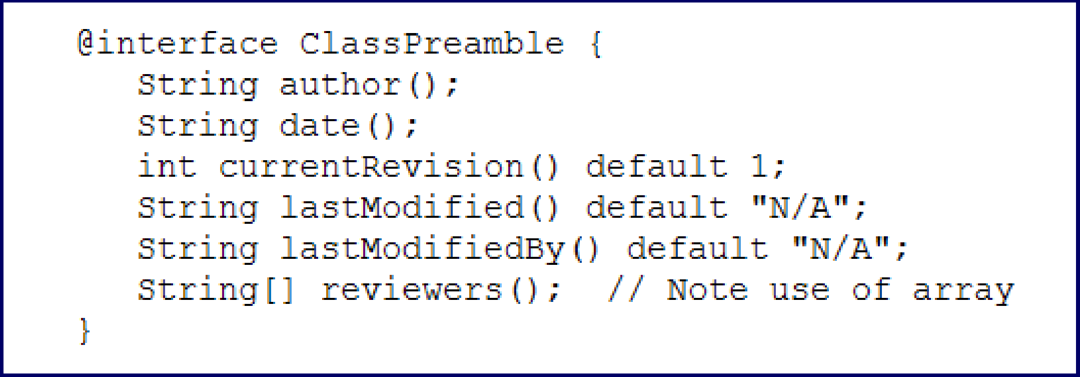
```
public class Generation3List extends Generation2List {

   // Author: John Doe
   // Date: 3/17/2002
   // Current revision: 6
   // Last modified: 4/12/2004
   // By: Jane Doe
   // Reviewers: Alice, Bill, Cindy

   // class code goes here

}
```

vs.

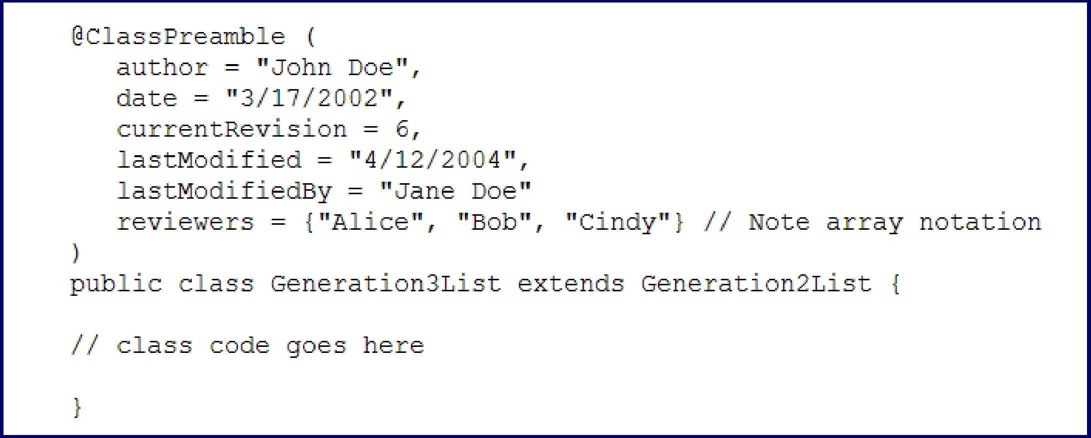
- Annotation

```
@ClassPreamble (
   author = "John Doe",
   date = "3/17/2002",
   currentRevision = 6,
   lastModified = "4/12/2004",
   lastModifiedBy = "Jane Doe"
   reviewers = {"Alice", "Bob", "Cindy"} // Note array notation
)
public class Generation3List extends Generation2List {

// class code goes here

}
```

Helpers for agile methodology: Code Annotation

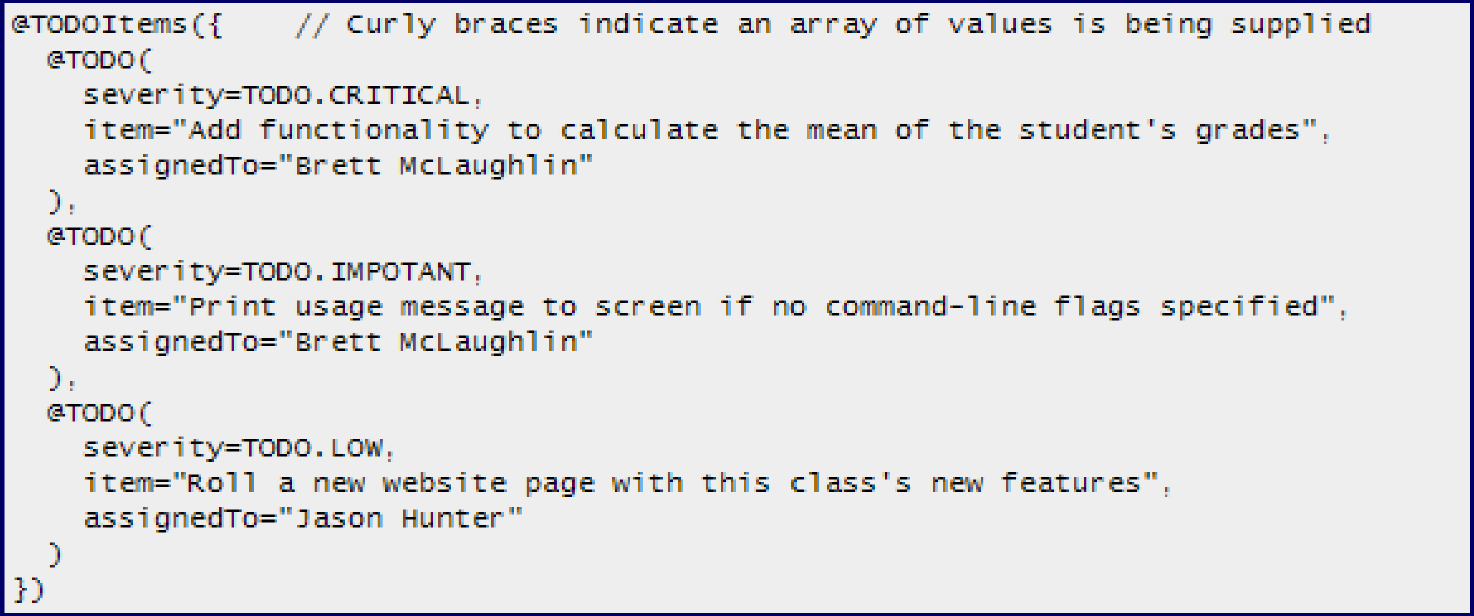


```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
   String[] reviewers();  // Note use of array
}
```



```
@ClassPreamble (
   author = "John Doe",
   date = "3/17/2002",
   currentRevision = 6,
   lastModified = "4/12/2004",
   lastModifiedBy = "Jane Doe"
   reviewers = {"Alice", "Bob", "Cindy"} // Note array notation
)
public class Generation3List extends Generation2List {

// class code goes here

}
```

Helpers for agile methodology: Code Annotation



```
@TODOItems({    // Curly braces indicate an array of values is being supplied
  @TODO(
    severity=TODO.CRITICAL,
    item="Add functionality to calculate the mean of the student's grades",
    assignedTo="Brett McLaughlin"
  ),
  @TODO(
    severity=TODO.IMPOTANT,
    item="Print usage message to screen if no command-line flags specified",
    assignedTo="Brett McLaughlin"
  ),
  @TODO(
    severity=TODO.LOW,
    item="Roll a new website page with this class's new features",
    assignedTo="Jason Hunter"
  )
})
```