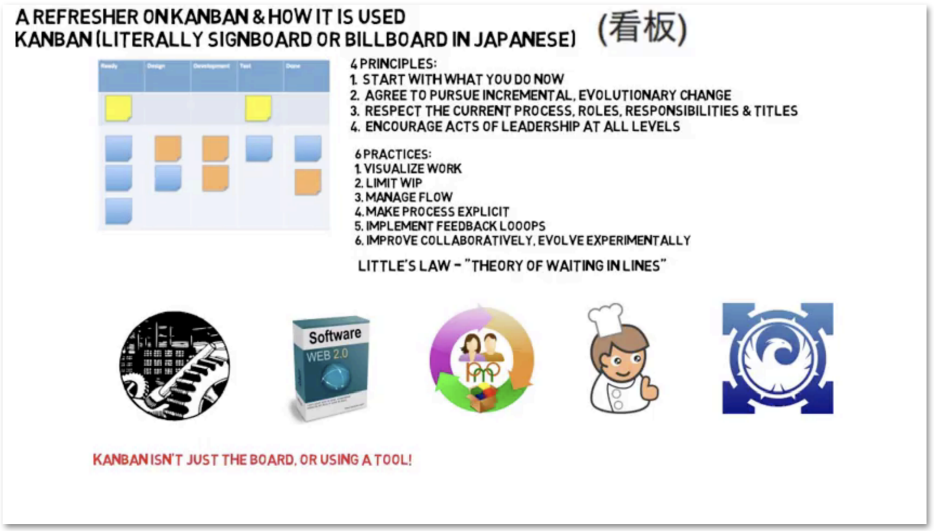
Kanban — Starting “Simple”



Timeboxing in Scrum

Every event in Scrum is timeboxed. Below are some examples for each, bearing in mind that every Scrum project is planned differently, and therefore different periods of time apply.

Sprint - Between 1 and 4 weeks

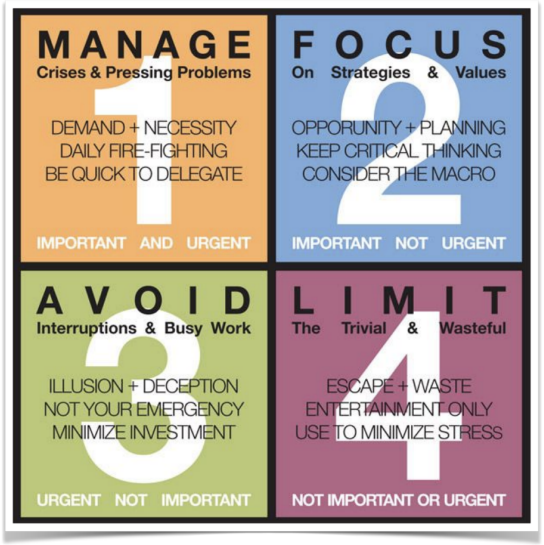
Sprint Planning - 2 hours for each week in a Sprint

Daily Scrum - 15 minutes

Sprint Review - 2 to 4 hours

Sprint Retrospective - 60 to 90 minutes

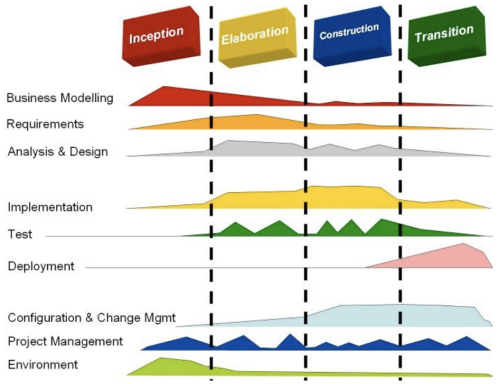
Evolution of Time Management



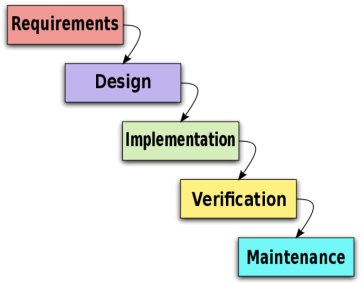
Value Driven

Unified Process vs. The Waterfall

Unified Process, aka UP. Also, RUP, the more well known cousin



Business value is delivered **incrementally** in time-boxed cross-discipline **iterations**



Big “**Design up front**”. The **Phase Gate exit**.

Agile Principles

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	7	Working software is the primary measure of progress.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	9	Continuous attention to technical excellence and good design enhances agility.
4	Business people and developers must work together daily throughout the project.	10	Simplicity--the art of maximizing the amount of work not done--is essential.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	11	The best architectures, requirements, and designs emerge from self-organizing teams.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Values of Extreme Programming

Extreme Programming

Extreme Programming (XP) is based on values. The rules we just examined are the natural extension and consequence of maximizing our values. XP isn't really a set of rules but rather a way to work in harmony with your personal and corporate values. Start with XP's values listed here then add your own by reflecting them in the changes you make to the rules.

Simplicity: We will do what is needed and asked for, but no more. This will maximize the value created for the investment made to date. We will take small simple steps to our goal and mitigate failures as they happen. We will create something we are proud of and maintain it long term for reasonable costs.

Communication: Everyone is part of the team and we communicate face to face daily. We will work together on everything from requirements to code. We will create the best solution to our problem that we can together.

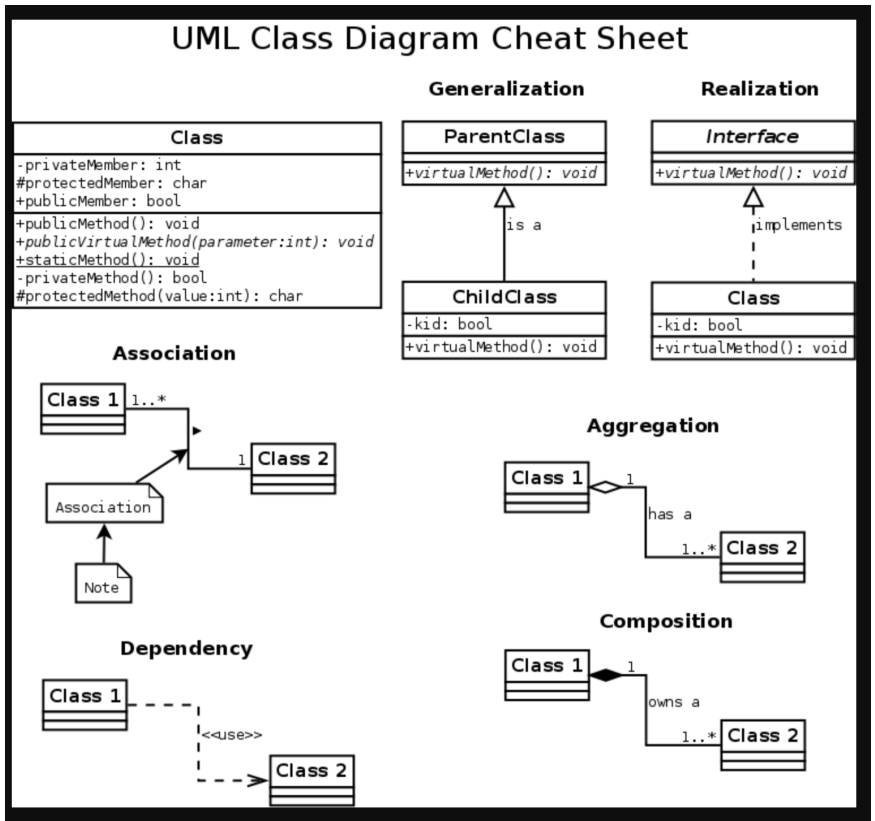
Feedback: We will take every iteration commitment seriously by delivering working software. We demonstrate our software early and often then listen carefully and make any changes needed. We will talk about the project and adapt our process to it, not the other way around.

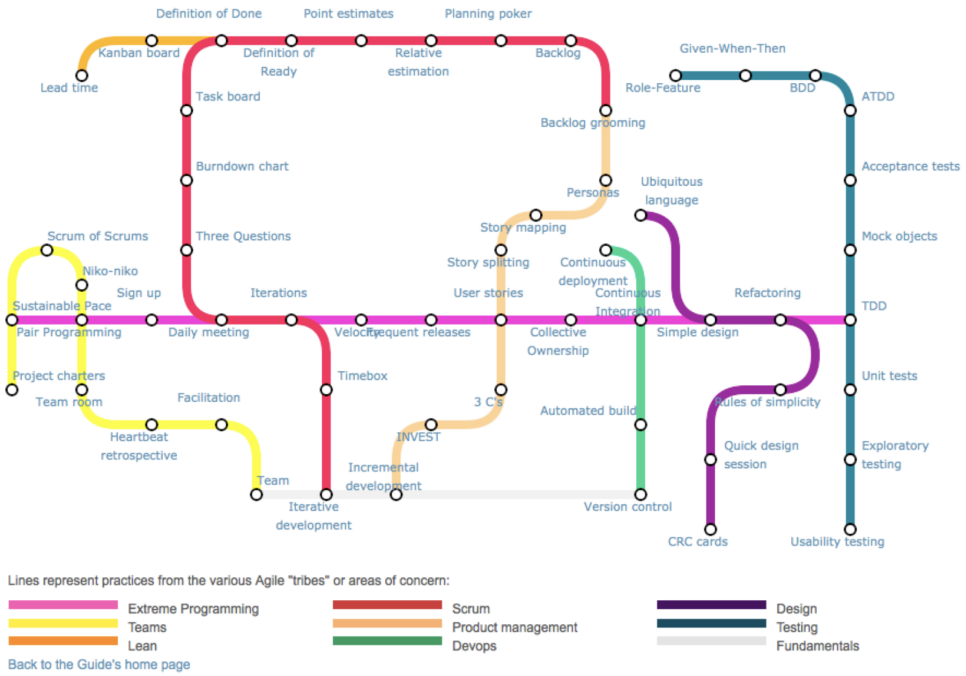
Respect: Everyone gives and feels the respect they deserve as a valued team member. Everyone contributes value even if it's simply enthusiasm. Developers respect the expertise of the customers and vice versa. Management respects our right to accept responsibility and receive authority over our own work.

Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes when ever they happen.

What lessons have we learned about implementing XP so far.

UML Class Diagram Cheat Sheet





The Rules of Extreme Programming

Planning

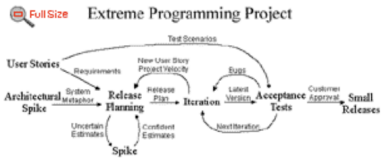
- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.



Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

The Values of Extreme Programming

Extreme Programming (XP) is based on values. The rules we just examined are the natural extension and consequence of maximizing our values. XP isn't really a set of rules but rather a way to work in harmony with your personal and corporate values. Start with XP's values listed here then add your own by reflecting them in the changes you make to the rules.

Simplicity: We will do what is needed and asked for, but no more. This will maximize the value created for the investment made to date. We will take small simple steps to our goal and mitigate failures as they happen. We will create something we are proud of and maintain it long term for reasonable costs.

Communication: Everyone is part of the team and we communicate face to face daily. We will work together on everything from requirements to code. We will create the best solution to our problem that we can together.

Feedback: We will take every iteration commitment seriously by delivering working software. We demonstrate our software early and often then listen carefully and make any changes needed. We will talk about the project and adapt our process to it, not the other way around.

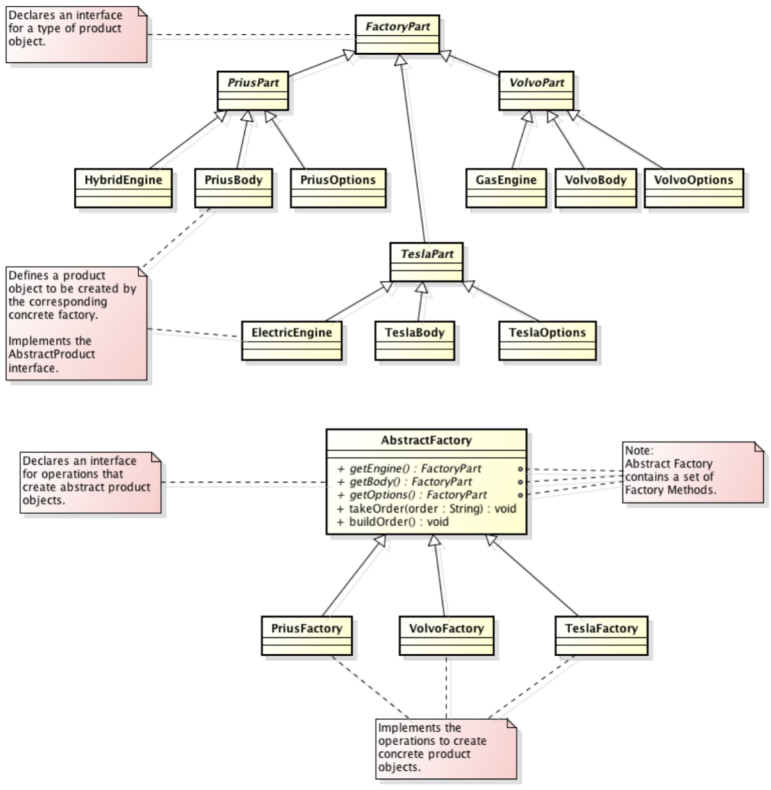
Respect: Everyone gives and feels the respect they deserve as a valued team member. Everyone contributes value even if it's simply enthusiasm. Developers respect the expertise of the customers and vice versa. Management respects our right to accept responsibility and receive authority over our own work.

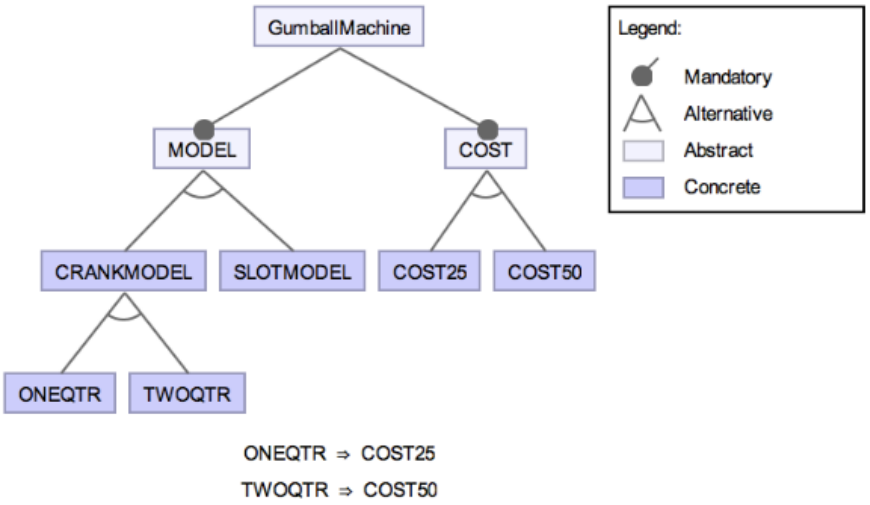
Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes when ever they happen.

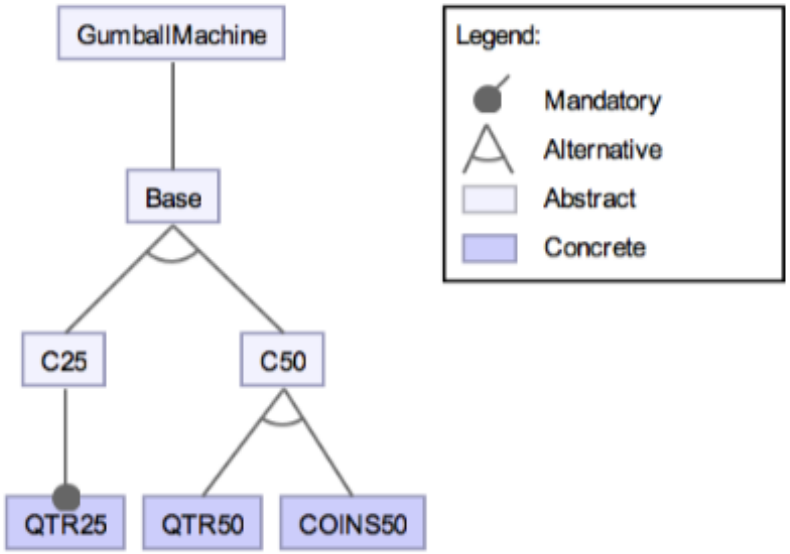
What lessons have we learned about implementing XP so far.

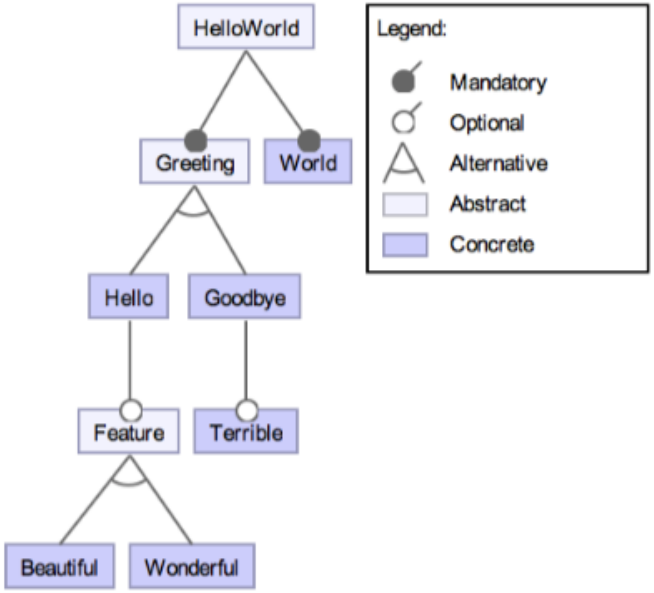
ExtremeProgramming.org home | XP Rules | XP Map | Lessons Learned | About the Author


```
public aspect TracingAspect {
	private int callDepth;

	pointcut traced() : !within(TracingAspect) && execution(public * *.*(..)) ;
	//pointcut traced() : !within(TracingAspect) && execution(* *.*(..)) ;

	before() : traced() {
		print("Before", thisJoinPoint);
		callDepth++;
	}

	after() : traced() {
		callDepth--;
		print("After", thisJoinPoint);
	}

	private void print(String prefix, Object message) {
		for (int i = 0; i < callDepth; i++) {
			System.out.print(" ");
		}
		System.out.println(prefix + ": " + message);
	}
}
```

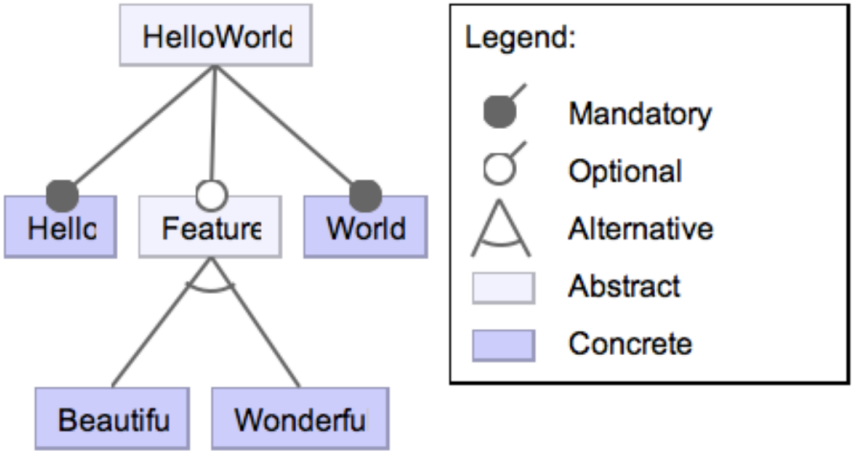








Hello World Feature Model



AspectJ 5 Quick Reference

Aspects *at top-level (or* **static** *in types)*

aspect *A* { … }
defines the aspect *A*
privileged aspect *A* { … }
A can access private fields and methods
aspect *A* **extends** *B* **implements** *I, J* { … }
B is a class or abstract aspect, *I* and *J* are interfaces
aspect *A* **percflow**(*call(void Foo.m())*) { … }
an instance of *A* is instantiated for every control flow through calls to *m()*

AOP & Databases

AOP

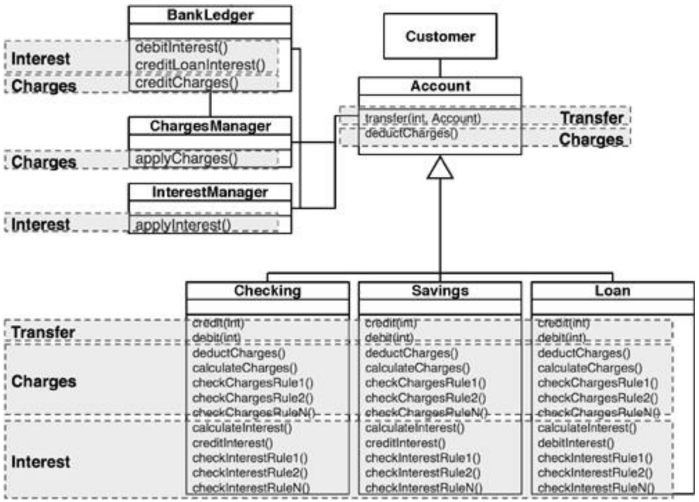
- Joinpoint
  - Before execution of method
  - After execution of method
- Pointcut
  - Select all classes
- Advice
  - Behavior added at joinpoint
- Introduction
  - Create classes out of the assembly of parts (i.e. fields, methods, etc…)

Databases

- Triggers
  - Before insert
  - After update
- Query
  - Select from tables
- Active Elements
  - Behavior fired at trigger
- Views
  - Create new "virtual" tables out of existing ones

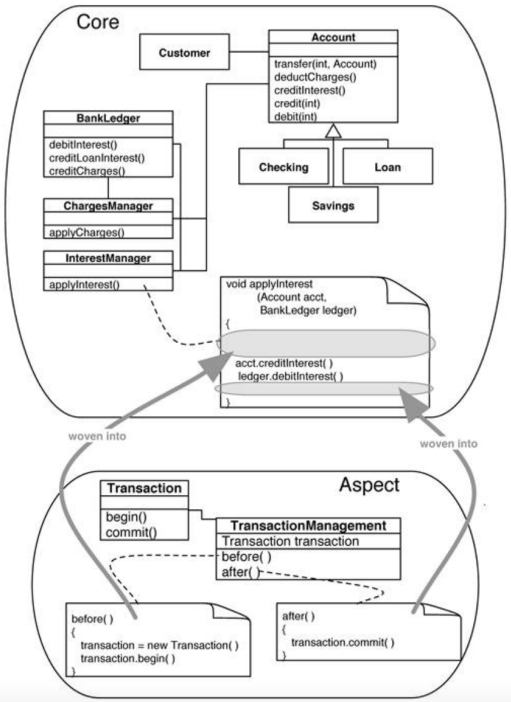
Symmetric Aspects (cont.)

Cross-Cutting Concerns in Core



Asymmetric Aspect (cont.)

Cross-Cutting Concerns Are Modularized as "Aspects"



Asymmetric Aspects

Crosscutting
: Concern behavior that is triggered in multiple situations.

Advice
: The triggered behavior.

Aspect
: The encapsulation of the advice and the specification of where the advice is triggered.

Core
: The traditional object-oriented part of the system to which aspects are applied.

Joinpoint
: A possible execution point that triggers advice.

Pointcut
: A predicate that can determine, for a given joinpoint, whether it is matched by the predicate

Weaving
: Applying the advice to the core at the joinpoints that match the pointcut statements in the aspects.