



# Agile Processes: Developing Your Own Secret Recipes

**Peter Coad**

Chief Strategist and SVP

June 17, 2003

**Borland**<sup>®</sup>  
Excellence Endures

# Purpose and agenda

- **Purpose**

- To view “process” from a new perspective
- To begin to mix-and-match mini-recipes from leading processes, so that your team might be better equipped to build better software faster

- **Agenda**

- Part 1: On the importance of recipes
- Part 2: Overall recipes
- Part 3: Mini-recipes

# In search of better recipes

## ▪ **Waterfall**

- Phases: require, analyze, design, develop, test
- Assumes: requirements can be both “known in advance” and “frozen”

## ▪ **Waterfall variations**

- Waterfall with overlap
- Waterfall with subprojects
- Waterfall with risk management

## ▪ **Spiral**

- Start with a smaller scope, then broaden the scope with each iteration. For each iteration:
  - 1. Determine objectives, alternatives, and constraints
  - 2. Identify and resolve risks
  - 3. Evaluate alternatives
  - 4. Develop deliverables for this iteration; verify correctness
  - 5. Plan the next iteration
  - 6. Commit to an approach for the next iteration

## ▪ **Unified Process, Feature-Driven Development, Extreme Programming**



Part 1  
**On the Importance  
of Recipes**

# Coffee cake

Here is a recipe, consisting of two mini-recipes

## Coffee cake

recipe

### ▪ Batter

mini-recipe

- 2 C baking mix
- ¼ C brown sugar or honey
- 1 egg
- 2/3 C milk or water
- 2 to 3 T melted butter or oil
- Beat together sugar, egg, milk, melted butter, and stir into baking mix until just moistened. Fold in nuts, if used. Spread batter into greased 9" layer pan or 8" square pan.

### ▪ Topping

mini-recipe

- 2 T brown sugar
- ¼ C baking mix
- 1 t cinnamon
- 1 T butter
- In a small bowl, mix topping ingredients until crumbly. Sprinkle over batter.

### ▪ Bake

- At 400 degrees, 20 to 25 minutes

# Recipes and mini-recipes

- **Select your (overall) recipe.**
  - The one you use now
  - An off-the-shelf starting point
    - Unified Process (UP)
    - Feature-Driven Development (FDD)
    - eXtreme Programming (XP)
  - A custom blend
- **Select your mini-recipes and add them in.**
  - Four mini-recipes: define, design, develop, test
  - For each mini-recipe...
    - Bring in the steps that best fit your team, your project, your challenges—that is to say, the ingredients on hand!
    - Select or blend practical how-to steps from your current process, UP, FDD, or XP.

# A spectrum of recipes

Different ingredients, costs, time to produce, and results



## • Chocolate Decadence Cookies

2 cups semi-sweet chocolate chips  
6 oz unsweetened chocolate  
12 Tbsp butter  
2/3 cup flour  
1/2 tsp baking powder  
1/2 tsp salt  
6 eggs  
2 cups sugar  
4 tsp vanilla

–Melt semi-sweet chips, unsweetened chocolate & butter in microwave. Stir until smooth. Sift flour, baking powder and salt together. In large mixing bowl, beat eggs, sugar and vanilla until fluffy. Beat in chocolate, and then flour. Stir in nuts. Drop by tablespoons onto lightly greased sheet. Bake 10-12 minutes at 350.

## • Chocolate Chip Cookies

Preheat oven to 350.  
Cut cookie dough into 1/2 inch slices.  
Bake 10-12 minutes until done.

# A spectrum of recipes



# You compete with your recipes!

- **You follow recipes today.**
  - Written or not
- **You compete with your recipes today.**
  - You win with your recipes.
  - You can better compete against others by studying their recipes.
- **You can increase your competitive edge.**
  - By improving your overall recipes
  - By improving mini-recipes within the recipes



## Part 2

# Overall Recipes

# Overall recipe—UP

- **Major emphasis on up-front specification**
  - Two specification phases: inception, elaboration
    - With define (two parts: requirements, analysis) and design, sometimes develop and test—within the inception phase
    - With define, design, develop, and test—within the elaboration phase
- **Followed by large-scale code production**
  - Two implementation phases: construction, transition
    - With define, design, develop, and test—within the construction phase
    - With develop and test—within the transition phase

# Overall recipe—FDD

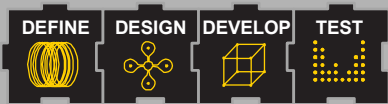
- **Modest amount of time and effort up-front**
  - Activities
    - Develop an overall model (10% up-front, 4% on-going).
    - Build a features list (2% up-front, 1% on-going).
    - Plan by feature (2% up-front, 2% on-going).
  - Roles
    - Model-and-plan teams (domain experts, modelers, chief architect, manager)
- **Many short design-develop-test iterations**
  - Activities
    - Design, develop, test—by feature—in “up to two-week” time boxes (77% of project time).
  - Roles
    - Feature teams (chief programmers, class owners)

# Overall recipe—XP

- **Very modest amount of time and effort up-front**
  - Vision card
  - Metaphor
  - User stories
  - Acceptance tests
- **Many short design-develop-test iterations**
  - Test first, test often
  - Pair programming
  - Community ownership of code
  - Refactoring
  - Continual integration



Part 3  
**Mini-recipes**  
Define, Design, Develop, Test



# Selecting mini-recipes

	UP	FDD	XP
Define	Use cases and class diagrams	Features list and class diagrams (subteams then teams)	User stories
Design	Sequence diagram	Sequence diagram	Refactor
Build	Code	Feature teams (chief programmers and class owners)	Pair programming (team ownership)
Test	Code, test, inspect	Code, test, inspect	Write tests, code, test—and continuously inspect

# Recommended reading

- **UP**
  - *UML and the Unified Process*  
Arlow and Neustadt
- **FDD**
  - *A Practical Guide to Feature-Driven Development*  
Palmer and Felsing
- **XP**
  - *A Practical Guide to Extreme Programming*  
Miller, Astels, and Novak
- **Spiral**
  - *Rapid Development: Taming Wild Software Schedules*  
McConnell

# Summary and conclusion

- **Yes...**
  - Your company competes with its recipes
- **Hence, getting started...**
  - *Agree* upon your overall recipe.
  - *Add* mini-recipes of your choosing.
  - *Do*, within your teams, on your projects
  - *Seek to improve, all along the way*
    - How is your project doing?
    - What are your project pain points?
    - What mini-recipes *within IT* could your project adopt and adapt?
    - What mini-recipes *outside IT* could your project adopt and adapt?



# Agile Processes: Developing Your Own Secret Recipes

**Peter Coad**  
Chief Strategist and SVP

**Borland**<sup>®</sup>  
Excellence Endures

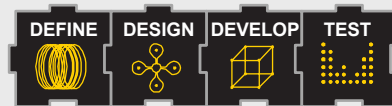


Appendix  
**Mini-recipes**  
Define, Design, Develop, Test

# Mini-recipes

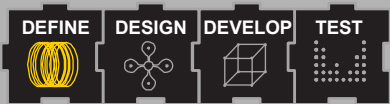
- **Four development mini-recipes (this presentation focuses on these four)**

- Define
- Design
- Develop
- Test



- **Additional mini-recipes**

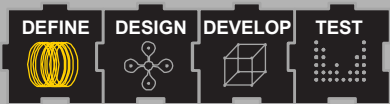
- Program management, project management, business-process modeling, configuration management, development environment, release management



# Define mini-recipe—UP

1 of 2: require

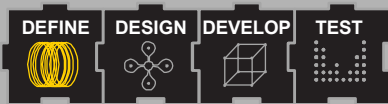
Role	Steps
<b>Requirements engineer</b>	<p><b>Establish (elicit, discover, and specify) requirements.</b></p> <p><b>Establish functional requirements.</b></p> <p>Template: &lt;reqt id&gt; &lt;system acronym&gt; shall &lt;function&gt;.</p> <p>Template: &lt;reqt id&gt; &lt;system acronym&gt; shall output &lt;output name&gt;, based upon this function or one similar to it: &lt;function details go here&gt;.</p> <p><b>Establish non-functional requirements.</b></p> <p><b>Prioritize requirements.</b></p> <p><b>Establish a project glossary.</b></p>
<b>System analyst</b>	<p><b>Establish use cases.</b></p> <p><b>Find the system boundary.</b></p> <p><b>Find actors.</b></p> <p><b>Find use cases; specify them; write some scenarios.</b></p> <p><b>Categorize and prioritize use cases.</b></p> <p><b>Trace requirements to use cases.</b></p>
<b>UI designer</b>	<p><b>Prototype user interfaces.</b></p>



# Define mini-recipe—UP

2 of 2: analyze

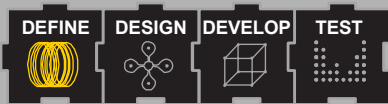
Role	Steps
<b>System analyst</b>	<p><b>Analyze the use cases.</b> Analyze noun-verb pairs within the use cases: look for class-method pairs, attribute-method pairs, and result-method pairs.</p> <p><b>Add classes.</b> Add problem-domain classes. Add 3-5 methods, 3-5 attributes, and associations.</p> <p><b>Add interactions.</b> Work out sequence diagrams for several methods.</p> <p><b>Organize into packages.</b> Put together packages of cohesive, semantically-related use cases, class diagrams, and sequence diagrams.</p>
<b>Architect</b>	<p><b>Define the architecture.</b></p>



# About UP's use cases

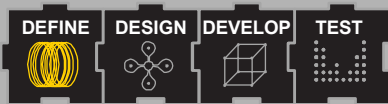
*The best use case diagrams are **simple**.  
Effective communication of functional requirements is key.*

- **Actor**
  - A role (or combination of roles) that some person, place, or thing adopts when directly interacting with the system. If it simplifies matters: organize by roles and combinations of little roles; only after that organize by kinds of roles.
- **Use case**
  - A behavior or combination of behaviors that the system exhibits to benefit one or more actors. What the system does for each actor; how each actor uses the system. If it simplifies matters: compose use cases of smaller ones, using <include> links; adapt an existing use case in defined ways using <extend> links. If it simplifies matters: organize by activities and combinations of little activities; only after that organize by kinds of activities.
- **Use case specification**
  - List preconditions, primary (and when significant, secondary) scenarios, and postconditions.
- **System boundary**
  - An initial understanding regarding what is in and what is out, evolves over time
- **Requirements traceability**
  - Using a table, trace use cases back to a separate requirements document (two expressions of requirements; not very efficient; yet sometimes necessary sociologically).



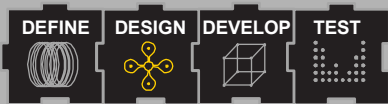
# Define mini-recipe—FDD

<p><b>Model-and features team (chief architect, modelers, domain experts)</b></p>	<p><b>Develop an overall model.</b>          Use the “subteam then team” process (subteams model in parallel several hours, then work together to produce an even better result).          Use the “domain-neutral component” (a template for building better object models).          Add 3-5 methods, 3-5 attributes, and associations to each class.          Work out sequence diagrams for several methods, especially methods that stretch the model, increasing understanding and improving content.</p>
<p><b>Model-and-features team</b></p>	<p><b>Build a features list.</b>          Client-valued functionality which can be designed and built in two weeks or less          Use this template: &lt;action&gt; &lt;result&gt; &lt;object&gt;          Example: Calculate the total of a sale.          Organize into features, feature sets, and major feature sets.          In FDD: the features list <u>is</u> the requirements spec.</p>
<p><b>Model-and-features team</b></p>	<p><b>Develop a feature plan.</b>          Factor-in dependency, priority, level of effort, technical risk, and schedule risk.</p>
<p><b>Model-and-features team</b></p>	<p><b>Inspect: checklist-driven walkthroughs</b>          Inspect the model, the list, and the plan</p>



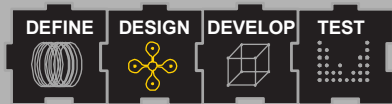
# Define mini-recipe—XP

<b>Customer</b>	<b>Vision card: system purpose, in 25 words or less</b>
<b>Analogy thinker</b>	<b>Metaphor: a way relate what the system is about to others “What we are building is in certain key ways like a...”</b>
<b>Storyteller and listeners</b>	<b>User stories (one per card, organized into stacks) To the customer (who is also an end-user) on the team: 1. Tell us stories about what you’d like to accomplish. 2. Then let’s adapt the stories, with software doing its part. 3. Then let’s divide the software stories into “design-and-build in hours or days” stories.</b>
<b>Team</b>	<b>Acceptance tests (one per card) A user story describes a general behavior. An acceptance test is a concrete situation in which that behavior is exercised. Three parts: setup, operation, verification</b>
<b>Team</b>	<b>A note on decision-making responsibility 1. Business people set scope, priorities, and release dates. 2. Technicians estimate, trade-off approaches, own their process, and set detailed schedules.</b>



## (b) Design mini-recipe—UP

<b>Architect</b>	<p><b>Design subsystems.</b> Example: problem domain, user interaction, data management</p> <p><b>Design subsystem interaction with sequence diagrams.</b></p> <p><b>Design the architecture of the system.</b></p> <p><b>Design major components and layers.</b></p> <p><b>Design communication and coordination mechanisms.</b></p> <p><b>Design deployment of the system.</b></p> <p><b>Design how the software will be distributed across available computational resources.</b></p>
<b>Designer</b>	<p><b>Add design content.</b> What? Add just enough detail to make it ready-to-implement. Where? Build and maintain separate analysis and design models, or just one model, or just one model with view management to extract-out the analysis model.</p> <p><b>Design use cases.</b></p> <p><b>Design classes and interfaces.</b></p> <p><b>Add detail with state diagrams, when useful.</b></p> <p><b>Design sequence diagrams.</b></p>

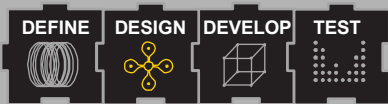


# Design mini-recipe—FDD

**44% of a “design by feature, build by feature” iteration**

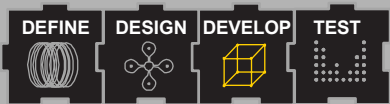
(two weeks or less of effort; elapsed time could be more than two weeks, depending upon how many features you are attacking in parallel and the amount of overlap between those features)

<b>Chief programmer</b>	<p><b>Form a feature team.</b>  <b>Who:</b> class owners, the ones needed for this feature  <b>How long:</b> Team exists for 2 weeks or less.  <b>Why class owners:</b> conceptual integrity + individual responsibility, authority, and accountability + familiarity  <b>Separation of concerns:</b> problem domain, user interaction, data management</p>
<b>Feature team</b>	<p><b>Conduct a domain walkthrough.</b>          If feature complexity warrants it.</p>
<b>Feature team</b>	<p><b>Study the referenced documents.</b></p>
<b>Feature team</b>	<p><b>Develop sequence diagram(s) for this feature.</b></p>
<b>Feature team</b>	<p><b>Conduct a design inspection.</b></p>
<b>Chief programmer</b>	<p><b>Refine the overall model.</b>          Feedback loop, continual improvement</p>



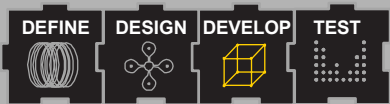
# Design mini-recipe—XP

Role	Steps
<b>Team</b>	<b>Participate in daily standup meetings</b> (no chairs). Pairs share: what we did yesterday, what we'll do today, and open issues. Short, to the point, and go!
<b>Pair</b>	<b>Write tests (design with tests) together.</b> Each test defines desired behavior.
<b>Pair</b>	<b>Refactor code (design in code) together.</b> See it three times? Refactor. Comment “how” rather than “why?” Refactor. Feels needlessly complex? Refactor.



## (c) Develop mini-recipe—UP

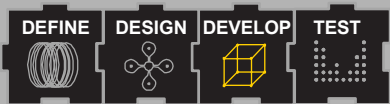
Role	Steps
Programmer	Implement classes and subsystems. Write and run unit tests.
Infrastructure programmer	Implement architecture.
Integration team member	Integrate system components.



# Develop mini-recipe—FDD

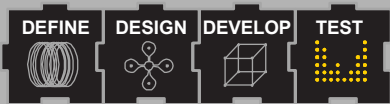
56% of a “design by feature, build by feature” iteration

Role	Steps
Feature team	Implement classes and methods.
Feature team	Conduct a code inspection.
Feature team	Do unit tests.
Feature team	Promote to build.



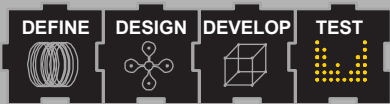
# Develop mini-recipe—XP

Role	Steps
Pair	<b>Write tests (design with tests) together.</b> Each test defines desired behavior.
Pair	<b>Write code (design in code) together.</b> Do the simplest thing that will work. Continuously simplify. Apply agreed-upon coding standards.
Pair	<b>Run tests together.</b>
Team	<b>Integrate results</b> ...into the production code base, throughout each day.



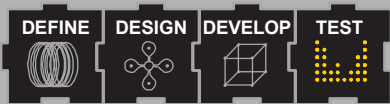
## (d) Test mini-recipe—UP

Role	Steps
<b>Programmer</b>	<b>Write and run unit tests.</b>
<b>Elaboration tester</b>	<b>Test the architectural baseline.</b>
<b>Construction tester</b>	<b>Test the initial operational capability.</b>
<b>Transition tester</b>	<b>Perform beta testing. Perform on-site acceptance testing.</b>



# Test mini-recipe—FDD

<b>Model-and-plan team, feature team</b>	<b>Inspections throughout define, design, and build</b>
<b>Feature team</b>	<b>Unit tests for each non-private class method</b>
<b>Test team</b>	<b>Integration tests running end-to-end, by feature. Do this in parallel, throughout the project.</b>
<b>Test team</b>	<b>System tests Test the system by feature sets. Do this in parallel, throughout the project.</b>
<b>Test team</b>	<b>Acceptance tests, in the production environment</b>
<b>Feature team</b>	<p><b>Measure and publish features complete.</b> Each feature is designed-and-built in 2 weeks or less. Design 4.4 days; build 5.6 days; very granular measures. Progress transparency: progress or the lack thereof is observable, measurable—and undeniable.</p> <p><b>Measure and publish the features-complete rate.</b> When the rate goes flat, fix it. When the rate jumps up, find out why and do more of it.</p>



# Test mini-recipe—XP

Role	Steps
Customer	Write acceptance tests.
Pair	Write tests first, for focus. Test for everything that could possibly break. Test before writing something new; test before you refactor; test after you refactor.
Team	Re-run all tests before and after integrating.
Team	Run acceptance tests. Use acceptance-test outcomes to measure progress.