Extreme Modeling

Embedded Systems Conference Chicago
June 3-6, 2002
Donald E. Stephens Convention Center
Chicago, Ill
Class 306 & 316

Extreme Modeling:
Part I

Stephen J. Mellor
Project Technology, Inc.
http://www.projtech.com

For more information, please visit our web site at http://www.projtech.com
Extreme Modeling

Context

Extreme Programming (XP)

Agile Alliance
- www.agilealliance.org

Executable Models
- See “Executable UML”

The Models are the Code
- Any specification that can be executed is “the code.”

Really?

Agile Manifesto

“We are uncovering better ways of developing software by doing it and helping others do it.

We value:
- <left> over <right>
- <left> over <right>
- <left> over <right>
- <left> over <right>"
Legend

Home: The principle in its short form.

Information: Explanation and more detail.

The Rant: Commentary

Agile Manifesto #1

*Individuals and interactions over processes and tools.*

You can build a system without processes and tools, but you can’t build a system without people.

And the better the people, the better the result. By far.

Lightweight processes are processes too!
Agile Manifesto #2

- **Working software over comprehensive documentation.**

- Models are not as valuable as real product, *unless* they can be executed.

**Confusion on the role of documents:**
- Aids
- Formal models
- Sketches.

Agile Manifesto #3

- **Customer collaboration over contract negotiation.**

- Tradeoffs are the “customer’s” responsibility.

- Development estimates are the developers’.

**Interaction is required to find the biggest bang for the buck in terms of:**
- Features
- Hardware

For more information, please visit our web site at [http://www.projtech.com](http://www.projtech.com)
Extreme Modeling

Agile Manifesto #4

Responding to change over following a plan.

Achieving the plan milestones does not necessarily relate to customer success. Adaptive vs. Predictive approaches

The target changes quickly with:
- The market
- The technology

Reality provides real feedback!

#1: Deliver Value

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”

Early and continuous delivery builds trust.

A non-technical point!
#2: Harness Change

“Welcome changing requirements, even late in development. Agile processes harness change for customers’ competitive advantage.”

Feedback, to be meaningful, implies change.
We should not resist change, but use it.

Models, like programs, can be made resistant to change, by modeling the invariant.

#3: Frequent Releases

“Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.”

“Delivery” is not always the same as “release.”

A “delivery” is internal, to the team, but a “release” is external, to the client.

Executable models are working software.
Agile Alliance Principles: #3

Separation of application from architecture

Executable UML models

Translation according to rules

Code for the System

#4: Customer on Site

“Business people and developers must work together daily throughout the project.”

(IT) Business people = (RT) Experts + Marketing

Features change because the market changes and because of technology limitations.

Make yourself useful to the hardware engineers, operators, and marketing department!
#5: Trust Motivated Teams

“Build projects around motivated individuals, give them the environment and support they need, and trust them to get the job done.”

People have the first order effect. Trust them.
The team leader’s/manager’s job is to ease the work of the project, remove barriers etc

Programmers of the world unite!
Today’s developers know more about today’s technology than their managers.

#6: Face-to-Face

“The most efficient and effective method of conveying information with and within a dev’t team is face-to-face conversation.”

Break down the barriers between developers.

What are the right documents?
What do the right documents contain?
What about geographically distributed teams?
Extreme Modeling

#7: Working Software

“Working software is the primary measure of progress.”

Better to know you’re in trouble early than late.

Executable models are working software.

Verify the models.

Very open to misinterpretation — by hackers or management.

#8: Sustainable Development

“Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.”

Discourages relying on heroes, but doesn’t preclude sprints.

Removing errors made late the previous night.

This principle is a cultural statement.

What does your organization expect?

What do you expect? Stock options?
#9: Technical Excellence

“Continuous attention to technical excellence and good design enhances agility.”

The better the design, the easier it is to maintain.
A plea not to be “quick and dirty”

Refactor models as you would code.

#10: Simplicity is Essential

“Simplicity—the art of maximizing the amount of work not done—is essential.”

Simple approaches are easier to change.
YAGNI.
Minimalism.

An aside: XP explicitly assumes that code is easy to change (see #9)
#11: Self-Organizing Teams

"The best architectures, requirements, and designs emerge from self-organizing teams."

Iterative development.
Emergent properties.
Many interactions; few process rules.

Build teams from the skills you have.
Then fill in the gaps. Emergent?

#12: Reflect and Tune

"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

Process check—applies the principles recursively.

Adapt or die.
Extreme Modeling

XP

A. The Planning Game (cf. #4 Customer On Site (one part))
B. Short Releases (cf. #3 Frequent Releases)
C. Metaphor
D. Simple Design (cf. #10 Simplicity is Essential)
E. Testing
F. Refactoring (cf. ≈ #9 Technical Excellence)
G. Pair Programming (cf. #5 Trust Motivated Teams; #6 Face-to-Face; #11 Self-Organizing Teams.)
H. Collective Ownership (cf. #5, #6, #11)
I. Continuous Integration (cf. ≈ #7 Working Software)
J. 40-hour Week (cf. #8 Sustainable Development)
K. On-Site Customer (cf. #4 Work Together (one half))
L. Coding Standards

Metaphor

A system’s architecture can be described by metaphor.

“Like” a controller. (i.e. Monitor and Control)
“Like” a telephone. (i.e. Transporter)
“Like” a bank. (i.e. Transaction)

A metaphor can be written down!
**Testing**

Design and build the test cases first.

Increases confidence in the code base. Guarantees that what you have does work.

Executable models can be interpreted with a verifier.

---

**Pair Programming**

Program in pairs.

One person is coding (micro), while another is ruminating on the abstractions (macro).

Analysis and modeling teams of three or four are most effective.

Gatherer, Analyst, Modeler, Organizer roles.
Collective Ownership

The code belongs to all of us.

Encourages knowledge transfer.
Discourages shady practice!

Put the information and models in a repository.
Saves time finding information.

Communication Paths

\[ n(n-1)/2 = 6 \times 5/2 = 15 \]

Repository + n = 1 + 6 = 7

For more information, please visit our web site at http://www.projtech.com
### Coding Standards

Use defined coding standards.

Improves collective ownership and communication.

Applies equally to modeling. Important especially in Action Language.

---

### Just Enough

High assurance systems can require more intermediate outputs than advocated by agile processes.

Just enough:
- Documents
- Process
- Customer interaction

Don’t make it more efficient if you don’t need to do it at all!

Removing chaff can only improve productivity.
Extreme Modeling

Collective Ownership and Teams

Collective ownership enhances commitment.
- Building a repository for all information is the single largest improvement in productivity you can achieve.

Teams, and pairs, enhance productivity. They also improve quality—very important in high-assurance systems.
- Building constantly changing teams is the second most important thing you can do to increase productivity.

Adaptability

Agile processes are all about adapting, not predicting.

Plan to adapt:
- In response to technology limitations and changes
- To the market and end-customer needs
- The process itself
Documents and Documentation

Agile processes sometimes confuse:
- Sketches, masquerading as models with
- Models that *are* working software

You *can* document the metaphor.

---

Derived Requirements

Agile processes rely on a “customer” to determine which features to build next.

But many of our requirements are *derived*. 

For more information, please visit our web site at [http://www.projtech.com](http://www.projtech.com)
Context: Agile Process

Whatever process you have:
- Your team must embrace and own it.
- Your team must see value in it.
- It’s not working if the team views it as “overhead.”
- It’s not working if the customer doesn’t derive value.

…whether your team wants to be agile or not!

Context: Executable Modeling

Executable UML models
Separation of application from architecture
Translation according to rules
Code for the System
Context: Layers of Abstraction

The typical real-time system comprises multiple layers of abstraction.

Who is the “customer” for Logging?

How do we build a single “story”?

Thirteen Agile Principles

1. Business People on Site (a.k.a. “The Planning Game”)
2. Technical Experts on Site
3. Face-to-Face Communication
4. Trust Motivated, Self-Organizing Teams
5. Collective Ownership and Coding standards
6. Sustainable Development
7. Harness Change
8. Metaphors and Simple Designs
9. Technical Excellence and Refactoring
10. Testing
11. Continuous Integration with Frequent Releases
12. Estimate to Improve
13. Reflect and Tune (the process)

And so … **Deliver Value**
Business People on Site

Q: Who are “business people” in a real-time context?

A: Those who represent the application functionality, probably marketing.

Large amounts of work are required to support the application.

Business People on Site

Business people decide what features to build based on cost estimates provided by developers.

Decide what to build:
- continuously, and
- document it

(Can be features on cards.)

Maintain the cards!

Always maintain at least two piles:
- requirements for this release
- requirements for future releases
Tracking the Features

Alternatively, track the features using a Project Matrix:

- The steps in your method become the rows.
- The columns are the subsystems of the domains.

<table>
<thead>
<tr>
<th></th>
<th>Chemical Plant</th>
<th>AUI</th>
<th>Software Arch.</th>
<th>PIO</th>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Diagram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State Chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Features associated with logging

Technical Experts on Site

Requirements management is about:

- interviewing (subject matter) technical experts, and
- collating and organizing, often conflicting bits of information into a coherent specification.

It is also about:

- prioritizing the requirements with the business people
- the definition of the product
  - what it will do,
  - what it will not do,
  - and in which release it will do it
Face-to-Face Communication

Face-to-face communication is the highest bandwidth.

Geographical distribution requires other techniques:

- Best to have met the person, in person, at some time.
- Regular teleconferences.
- E-mail is a “pull” medium.
  - Use e-mail for technical stuff only.
  - Never use e-mail for emotional subjects.
  - Sleep on “difficult” e-mails.
- Focus on single task list to maintain momentum.

Build Motivated Teams

Start by determining the skills available:

- never start by enumerating the skills needed

Compare the skills you have to the skills you need:

- identify areas that are covered
- identify the gaps

Put people in the right jobs.

- figure out what each person does well
  - it's usually something that's enjoyed

Hire or contract the skills your team is missing.
Self-Organizing Teams

Encourage the team to:
- Recognize when a person is in the wrong job
  - unproductive, unhappy, meddling in other jobs
- Move people from wrong job to right job
  - should be seen as a positive thing by everyone
- Reassign or retrain people with unneeded or surplus skills.

But, hold teams accountable:
- plans, milestones, deliverables
- team morale, productivity, quality

Let the team tell you if something’s wrong.

Collective Ownership

To support collective ownership, projects require significant infrastructure:
- document library
- version control
- configuration management
- issue tracking
- development tools
Ownership: Artifact Library

Yes, you need an artifact library, now!

- well-known place to store work
- under version control
- directory system
  - artifact numbers and a catalog,
  - or
  - hierarchical, named directory structure

\[ n( n - 1 ) / 2 = 6 \times 5 / 2 = 15 \]

Ownership: Modeling Standards

Collective ownership means the artifacts look as if they were produced by a single mind.

This means standards:

- Model layout standards
- Naming standards
- Case and spacing standards
  (isEqual? IsEqual? Is_equal?)
- Action language standards
Sustainable Development

Sustainable development is more than just a 40hr week.

Provide proper (consulting) support:
- adopting any new technique requires effort
- guidance from experienced consultants is crucial
- allows team to focus on adding value

As estimation capability improves, sustainable hours become more, er, sustainable.

Harness Change

Don’t carve the initial requirements in stone.
- Requirements change, often;
  - deal with it!
- Requirements often conflict:
  - get used to it;
  - learn to prioritize

Changing requirements are your friends:
- Find them before they find you
- Accept them with enthusiasm
- Consider them as opportunities:
  - competitive edge
  - innovation
Extreme Modeling

Metaphors and Simple Designs

The analyst should have an ‘attitude’ of:

- finding the invariant in the problem domain
- expressing the specifics of the problem as data

The invariant is the physics of fluid transfer.

The plant is expressed in data.

Technical Excellence & Refactoring

Bad work results from:

- Working with incomplete information
- Delaying work that can be completed

Do no work before its time.

- Do work only when you have the information needed
- Do work only when it hurts!
- When you see improvements, make them.

For more information, please visit our web site at http://www projtech.com
Testing: Application Models

Test model compiler and application models independently.

- Test this for behavior as you build models
- Data-driven modeled test drivers using a verifier
- Test cases defined as pre-existing instances

Testing: Model Compilers

Test model compiler and application models independently.

- Test this for correctness and performance
- Execute application models and measure performance.
Testing: Target Model Compiler

Then, test application models using the target model compiler.

Frequent Releases

The domain chart is the framework for integration.

- Build the lowest level services first.
- Then integrate into its clients.
- And its clients……
- And its clients……

Build a “pipe” through the system to enable testing on the “real thing” as early as possible.

For more information, please visit our web site at http://www.projtech.com
### Metrics: Estimates

Collect only metrics that are either:
- definitely useful, or
- very cheap to collect

People will change their behavior to optimize metrics possibly making a negative impact on the end product.

So, be careful about:
- what you measure, and
- how measurements affect the team

#### Metrics: Estimation

Estimate relative sizes of the boxes.

<table>
<thead>
<tr>
<th>Relative column sizes</th>
<th>Chemical Plant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Diagram 1.00</td>
<td>Transfer 1.50</td>
</tr>
<tr>
<td>State Chart 1.50</td>
<td>Recipes 0.75</td>
</tr>
<tr>
<td>Actions 1.00</td>
<td>Control 1.25</td>
</tr>
</tbody>
</table>

The "reference box."

Relative row sizes
- Inventory 1.00
- Transfers 1.50
- Recipes 0.75
- Control 1.25
- Class Diagram 1.00
- State Chart 1.50
- Actions 1.00
- Map to Arch 0.50
- Screen Def’n 2.00
- Screen Files 0.25
- PIO Config 0.40
- Logging Config 0.20

For more information, please visit our website at [http://www.projtech.com](http://www.projtech.com)
Metrics: How to Use Them

Use metrics carefully to improve:
- estimation capabilities, and lead to more accurate plans
- alleviate the fears of your customers, and increase trust
- process
- team performance and confidence in plan
- individual performance

Reflect and Tune

- Be prepared to adjust the process as necessary.
- Any process that cannot be improved is destined to become obsolete.
- Encourage team members to suggest improvements.

Process check!
Thirteen Agile Principles

1. Business People on Site (a.k.a. “The Planning Game”)
2. Technical Experts on Site
3. Face-to-Face Communication
4. Trust Motivated, Self-Organizing Teams
5. Collective Ownership and Coding standards
6. Sustainable Development
7. Harness Change
8. Metaphors and Simple Designs
9. Technical Excellence and Refactoring
10. Testing
11. Continuous Integration with Frequent Releases
12. Estimate to Improve
13. Reflect and Tune (the process)

And so … Deliver Value

Brought to you by…

Translating Ideas Into Products.

Makers of BridgePoint ® and DesignPoint ® Modeling Tools

Stephen J. Mellor
Project Technology, Inc.
http://www.projtech.com

For more information, please visit our web site at http://www.projtech.com