THE CRYSTAL METHODS, OR
HOW TO MAKE A METHODOLOGY FIT

ALISTAIR COCKBURN
http://Alistair.Cockburn.us

HISTORY 1991 - 2004

1991: Alistair Cockburn told to develop an effective software development methodology.
- Interviewed and studied project teams for 10 years.
- Found that "people-centric methodologies" do better than "process-centric" methodologies.
- Found that you must choose and tailor the methodology to the team and the assignment (no methodology fits all projects).

1994: "Orange" used on 45-person fixed-price project
1997: "Orange" published in Surviving OO Projects
1998: Family of methodologies the name "Crystal"
2004: Crystal Clear published as book

PEOPLE LEARN SKILL IN A 3-STAGE PROGRESSION

Following - Breaking away - Fluency

Level 1: Following (Shu)
Learn "a technique that works"
"Success" is following the technique

Level 2: Breaking away (Ha)
Learn limits of the technique
Learn to shift from one technique to another

Level 3: Fluency (Ri)
Shift techniques by moment
Unable to describe the techniques involved

These apply to design, management, methodology, are relevant to project staffing.

7 PROPERTIES OF SUCCESSFUL PROJECTS

1. Frequent Delivery
2. Close/Osmotic Communication
3. Reflective Improvement
4. Personal Safety
5. Focus
6. Easy Access to Expert Users
7. Technical Environment with
   - Frequent integration
   - Automated testing
   - Configuration management

Core properties of "Crystal"
Properties increasing project safety
   (rubustness to adverse events)

1: Frequent Delivery
Have you delivered running, tested, usable functions to your user community at least twice in the last six months?

2: Reflective Improvement
Did you get together at least once within the last three months for a half hour, hour, or half day

   to compare notes, reflect, discuss your group’s working habits,
   and discover what speeds you up, what slows you down, and what you might be able to improve?
3: Close/Osmotic Communication

Does it take you 30 seconds or less to get your question to the eyes or ears of the person who might have the answer?

Do you overhear something relevant from a conversation among other team members at least every few days?

4: Personal Safety

Can you tell your boss you mis-estimated by more than 50 percent, or that you just received a tempting job offer?

Can you disagree with him or her about the schedule in a team meeting?

Can people end long debates about each other’s designs with friendly disagreement?

5: Focus

Do all the people know what their top two priority items to work on are?

Are they guaranteed at least two days in a row and two uninterrupted hours each day to work on them?

6: Easy Access to Expert Users

Does it take less than three days, on the average, from when you come up with a question about system usage to when an expert user answers the question?

Can you get the answer in a few hours?

7: Technical Environment with Automated Tests, Configuration Management, and Frequent Integration

Can you run the system tests to completion without having to be physically present?

Do all your developers check their code into the configuration management system?

Do they put in a useful note about it as they check it in?

Is the system integrated at least twice a week?

The Crystal Mindset

Software Development as a Cooperative Game
Making software consists only of making ideas concrete in an economic context:

People inventing and communicating, solving a problem they don’t yet understand (which keeps changing).

Creating a solution they don’t really understand (and which keeps changing).

Expressing ideas in restricted languages they don’t really understand, (and which keep changing)

To an interpreter unforgiving of error.

Resources are limited, and every choice has economic consequences.

*It is a cooperative game of invention and communication*

Software development is a Cooperative Game of Invention and Communication.

To understand team software development:

- Understand goal-directed cooperative games
- Understand people communicating
- Understand people inventing
- Understand people cooperating

Notes on Cooperative Game: Two goals:

Primary Goal → Deliver this software
Secondary Goal → Set up for the next game
Two conflicting games in one

Net result: Not repeatable!

Games, finite/infinite or cooperative/competitive, consist of better/worse ‘moves’

<table>
<thead>
<tr>
<th>Infinite</th>
<th>Finite w/ no fixed end</th>
<th>Finite &amp; goal-directed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Survival</td>
<td>King-of-the-hill wrestling</td>
<td>Tennis</td>
</tr>
<tr>
<td>Career Management</td>
<td>Poker</td>
<td>Rock-Climbing</td>
</tr>
<tr>
<td>Jazz music</td>
<td>Software Development</td>
<td></td>
</tr>
</tbody>
</table>

Every game run uses different strategies -- Set up each project suitably or suffer

```
<table>
<thead>
<tr>
<th>Life</th>
<th>Essential moneys</th>
<th>Discretionary moneys</th>
<th>Comfort</th>
</tr>
</thead>
<tbody>
<tr>
<td>L6</td>
<td>L20</td>
<td>L40</td>
<td>L100</td>
</tr>
<tr>
<td>L20</td>
<td>L40</td>
<td>L60</td>
<td>L200</td>
</tr>
<tr>
<td>L40</td>
<td>L60</td>
<td>L80</td>
<td>L500</td>
</tr>
<tr>
<td>L100</td>
<td>L200</td>
<td>L400</td>
<td>L1000</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>E200</td>
<td>E400</td>
<td>E600</td>
<td>E800</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>E500</td>
<td>E1000</td>
<td>E2000</td>
<td>E3000</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C6</td>
<td>C200</td>
<td>C400</td>
<td>C600</td>
</tr>
<tr>
<td>C200</td>
<td>C400</td>
<td>C600</td>
<td>C800</td>
</tr>
<tr>
<td>C400</td>
<td>C600</td>
<td>C800</td>
<td>C1000</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C1000</td>
<td>C2000</td>
<td>C4000</td>
<td>C6000</td>
</tr>
<tr>
<td>C2000</td>
<td>C4000</td>
<td>C6000</td>
<td>C8000</td>
</tr>
<tr>
<td>C4000</td>
<td>C6000</td>
<td>C8000</td>
<td>C10000</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

```

Naur 1986: The primary result of programming is the theory held by the programmers

1. Theory: The knowledge a person must have to do certain things intelligently, explain, answer queries, argue about them...
2. The programmer must Build a Theory of how certain affairs of the world will be handled by a program; Explain how the affairs of the world are mapped into the program and documentation; Respond to demands for modifications, perceiving the similarity of the new demand with the facilities built.
   - This knowledge transcends that possible in documentation.
3. This theory is the mental possession of a programmer; the notion of programmer as an easily replaceable component in program production has to be abandoned.

Naur 1986: Modifying a program depends on the new programmers building the same theory!

4. Problems of program modification arise from assuming that programming consists of text production, instead of theory building.
5. The decay of a program from modifications made by programmers without proper grasp of the underlying theory becomes understandable. The need for direct participation of persons who possess the appropriate insight becomes evident. For a program to retain its quality it is mandatory that each modification is firmly grounded in its theory.
6. The conclusion seems inescapable that at least with certain kinds of large programs, the continued adaption, modification, and correction, is essentially dependent on a certain kind of knowledge possessed by a group of programmers who are closely and continuously connected with them.
Perfect communication is impossible

You try to communicate what you “know”
• What you “know” depends on your individualized parsing of the world around you;
• You don’t know what it is you do know;
• You don’t know the thing you are trying to communicate;
• You don’t know what you are actually communicating;
• Your listener sees only a part of what you are saying;
• What your listener learns depends on his/her internal state.

(How is it we communicate at all?)

COMMUNICATION is touching into shared experience

Linked sequences of shared experience becomes a shared experience
➢ Project colleagues have rich shared experiences, a shortcut vocabulary

Implications for documentation:
➢ can never fully specify requirements
➢ can never fully document design
➢ must assume reader’s experiences
• more => can write less
• less => must write more.

Our task is to manage the incompleteness of communications

The Economics of Communication

Face-to-face allows vocal, subvocal, gestural information to flow, with fast feedback

People don’t ask questions if they have to climb stairs.

<table>
<thead>
<tr>
<th>Kim</th>
<th>Pat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Think (1/3 work year)</td>
<td>yr penalty ($300,000 / yr)</td>
</tr>
</tbody>
</table>

(1.5 work month) / yr penalty ($100,000 / yr)

(However, this fact makes for a useful strategy sometimes, ref: Cone of Silence)

Poor office layout costs the project a lot

Programmers cost = $ 2.10 / minute (± 50%)

Reference pair programming @ 100 questions/week
1 minute delay / question = (100 work min.)

... for 12 people, 12 months = 1.5 work months ($100,000)

+ Lost Opportunity Costs for questions not asked!
**Information drifts in currents -- (not unlike perfume)**

*Still Effective.*
- Nearby programming

*Most effective.*
- Programming in pairs


“Distance Matters,” Olson & Olson

---

**Notes: Put barriers up to reduce DRAFTS!**

Morale also flows through convection currents!

---

**Information radiators emit passively**

People get information just by walking past!

Photos courtesy of Thoughtworks

---

**Set up rooms to balance drafts and osmotic communication**

- Meeting
- Kitchen
- Programming work
- Private work
- Library

Watch for:
- drafts
- convection currents
- communities

---

**PEOPLE are essential but non-linear active components in the development process**

**Weak on:**
- Consistency
- Discipline
- Following instructions
- Changing habits

**Strong on:**
- Communicating
- Looking around
- Copy / modify

**Motivated by:**
- Pride in work
- Pride in contributing
- Pride in accomplishment
The alignment of people’s goals affects the team efficiency

A mission statement understood by all is critical to goal alignment.

Normal team

Aligned team

(Dirty Dozen again)

Amicability between people determines how quickly information moves

Amicability: Willingness to listen with good will

The “amicability index” indicates how easily information passes from one part of the organization to another.

A low amicability index implies that people block the flow of information, intentionally or through not listening well.

Amicability grows and rots fastest in osmotic communication settings!

People, cooperation, communication issues determine much of a project’s speed

Can they easily detect something needs attention? (Good at Looking Around)

Will they care enough to do something about it? (Pride-in-work; Amicability)

Can they effectively pass along the information? (Proximity; face-to-face, convection currents)

Methodologies as Transient

A methodology is a formula across people, but People change frequently

Revisit the team’s conventions every month

“Criticality”

Number of people coordinated

<table>
<thead>
<tr>
<th>“Criticality”</th>
<th>Life</th>
<th>Essential moneys</th>
<th>Discretionary moneys</th>
<th>Comfort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L6</td>
<td>E20</td>
<td>D60</td>
<td>C100</td>
</tr>
<tr>
<td></td>
<td>L120</td>
<td>E40</td>
<td>D200</td>
<td>C200</td>
</tr>
<tr>
<td></td>
<td>L400</td>
<td>E100</td>
<td>D500</td>
<td>C500</td>
</tr>
<tr>
<td></td>
<td>L1000</td>
<td>E1000</td>
<td>D1000</td>
<td>C1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of people coordinated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 6</td>
</tr>
</tbody>
</table>

©Alistair Cockburn 2003-7
How do we make methodology construction so cheap that we can construct them each month?

**Answer:** The Reflection technique

### Keep These

- Ongoing Problems

### Try These

Crystal’s “Genetic Code (DNA)”

Crystal is the lightest, least intrusive set of rules that puts a project in the safety zone.

**Crystal’s purpose:** Keep people from hurting each other, keeping each other informed

**Crystal’s nature:** A set of conventions that gets updated

**Crystal’s Philosophy:**
- People differ in working styles
- Projects differ in needs
- Software development is communication-intensive, experiment-based, needing lots of feedback in all directions
- Less is generally better (for methodologies)
- Techniques / technologies change over time

Crystal is a “family” because every project is slightly different.

Crystal’s common genetic code.

1. **Cooperative Game Mindset:**
   - A series of resource-limited cooperative games of communication and invention.

2. **Critical Project Properties:**
   - Frequent delivery
   - Close communication
   - Reflective Improvement

3. **Key Techniques:**
   - Discretionary, but with a starter set.

4. **Design Priorities:**
   - Project safety
   - Development efficiency
   - Habitability

5. **Design Principles:**
   - (7 principles, including:
     - face-to-face, concurrent development, different rules for different circumstances)

6. **Design Samples:**
   - Crystal Clear
   - Crystal Orange
   - Crystal Orange-web

The cooperative game has a primary and secondary goal: *Two Games in One!*

**Primary Goal**
- Deliver working software.
- (Mess up the first goal => no software.)

**Secondary Goal**
- Set up for the next game.
- Mess up the secondary goal => disadvantaged next project
**Crystal’s Project Properties**

- Frequent Delivery
- Osmotic Communication
- Reflective Improvement
- Personal Safety
- Focus
- Easy Access to Expert Users
- Technical Environment with
  - Frequent integration
  - Automated testing
  - Configuration management

---

**Timeboxing** is a critical element in iteration scheduling

2-week, 1-month (quarterly) timeboxes.

Each timebox ends with integrated, tested code.

Cut scope as needed but complete on time.

Deliver whatever you have

Whatever you accomplished this time is a predictor of what you will accomplish next time

("Yesterday’s Weather")

(some timeboxing fixes requirements, some don’t)

---

**Crystal’s Starter Strategies & Techniques**

- Exploratory 360°
- Early Victory
- Walking Skeleton
- Incremental Rearchitecture
- Information Radiators
- Methodology Shaping
- Reflection Workshop
- Blitz Planning
- Delphi Estimation
- Daily Stand-ups
- Agile Interaction Design
- Process Miniature
- Side-by-Side Programming
- Burn Charts

---

**Critical technique in Crystal:**

The *reflection workshop* each month or iteration.

Hang a 2-column flipchart

Fill in the chart (30 minutes)

Hang the chart in a public, visible, frequently seen place!

Try the ideas

Repeat each month or after each iteration
**Crystal’s Design Priorities**

- Project Safety
- Development Efficiency
- Process Habitability

**Crystal’s Design Principles**

1. Prefer face-to-face communication
   - Interactive face-to-face communication is the cheapest and fastest channel for exchanging information
2. Methodology weight is costly
3. Use heavier methodologies for larger / distributed teams
4. Use more ceremony for more criticality
5. Use more feedback & communications, with fewer intermediate deliverables
6. Discipline, skills, understanding counter process, formality, documentation
7. Efficiency is expendable at non-bottleneck activities.

**Crystal Sample Methodology Designs**

Crystal Orange  
Crystal Orange/web  
Crystal Clear

**Crystal Orange : scope**

For D40 projects:  
Up to 40 people, same building  
Loss of discretionary moneys  
(May extend to E50)

Not for very large projects  
(Insufficient subteaming)
Not for life-critical projects  
(Insufficient verification)

(Described in Surviving Object-Oriented Projects, Cockburn, 1998, pp. 77-83)

**Crystal Clear : scope**

For D6 projects:  
3-6 people, close or in same room  
Loss of discretionary moneys  
(may extend to E8 project)

Not for large projects  
(Insufficient group coordination)
Not for life-critical projects  
(Insufficient verification)

(Described in Crystal Clear, Cockburn, 2004  
also in Agile Software Development, Cockburn 2002)
Crystal Clear roles & teams for 3-8 people

**Required Roles:**
- sponsor,
- senior designer,
- designer/programmer,
- user (part-time)

**Combined Roles:**
- coordinator,
- business expert,
- requirements gatherer

**Teams:**
- single team of designer-programmers

**Seating:**
- single big room, or adjacent offices

---

Project Visibility

The “burn-down” chart and variations show a project’s progress visibly and publicly

Visibility of *rate-of-progress* and achievements are critical project management information

Works because task list is *fixed in size*

“Iceberg” list useful when task list changes daily
- Use spreadsheet or similar
- List tasks
- Developers estimate time, Managers prioritize
- Sort in priority order
- Derive which tasks are in schedule for this current iteration / delivery,
- Managers can change priorities

---

**Method #1. The whole house, by “importance”**

No view into progress & no warning of trouble

<table>
<thead>
<tr>
<th>% Complete</th>
<th>Surprise?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discover the 10% completely unexpected!</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Pack the 15% critical</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Pack the 80% not-so-critical</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Discard the 5% junk</strong></td>
<td></td>
</tr>
</tbody>
</table>

Works because task list is *fixed in size*

Method #2. Estimate boxes to be packed.

Good visible progress. No warning of trouble!

<table>
<thead>
<tr>
<th># Boxes Packed</th>
<th># Rooms Still to Pack</th>
</tr>
</thead>
<tbody>
<tr>
<td>(actual # boxes needed)</td>
<td>(actual # boxes needed)</td>
</tr>
<tr>
<td>expected # boxes needed</td>
<td>expected # boxes needed</td>
</tr>
<tr>
<td>slope = &quot;weezy&quot;!</td>
<td>slope = &quot;weezy&quot;!</td>
</tr>
<tr>
<td>An unknown number!</td>
<td>An unknown number!</td>
</tr>
<tr>
<td>Surprise!</td>
<td>Surprise!</td>
</tr>
</tbody>
</table>

Method #3. Packing each room to completion.

Good visibility & early warning.

This strategy works because:
1. The # rooms won’t increase.
2. Feedback is on full process.
Getting Started with Crystal

Select the frequency of delivery, the length of the iteration and integration cycles.

Focus on the first 3 properties

Must Do These!
1. Frequent Delivery: every month or two
2. Osmotic Communication: sit next to each other
3. Reflective Improvement: do reflection workshop monthly

Start work, stay in good-humored communication with your teammates!

Add these as you can ...
4. Personal Safety: speak without fear of punishment
5. Focus: Know what is critical, have time to work on it
6. Easy Access to Expert Users
7. Technical Environment with
   - Frequent integration: hourly, daily, 3 / week
   - Automated testing: unit tests, acceptance tests
   - Configuration management: check-in, versioning

Learn new techniques to get better ...

- Exploratory 360°
- Early Victory
- Walking Skeleton
- Incremental Rearchitecture
- Information Radiators
- Methodology Shaping
- Reflection Workshop
- Blitz Planning
- Delphi Estimation
- Daily Stand-ups
- Agile Interaction Design
- Process Miniature
- Side-by-Side Programming
- Burn Charts
- Test-First Design

Hold a reflection workshop each month.

Keep these
- test lock-down
- quiet time
- daily meetings

Try these
- pair testing
- fines for interruptions
- programmers help testers

Ongoing Problems
- too many interruptions
- shipping buggy code
• “Crystal” is a genetic code, shaping your working conventions to your project, always agile, focused on frequent delivery, close communication, & reflection.

• Crystal Clear is the lightest of the Crystal family, for 3-8 people working at the same location. It can be stretched to 18-20 people.

Read more at http://Alistair.Cockburn.us