# Crystal Methods and Spiral Lifecycle Model

An agile and a traditional software development methodologies

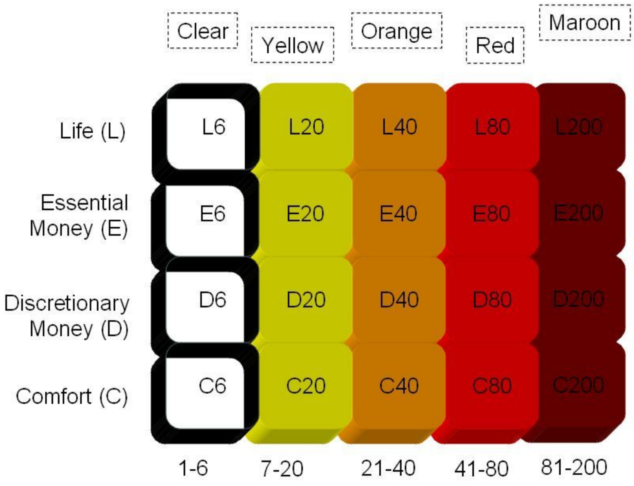
## Crystal Methods

Crystal Methods are a family of methodologies developed in the mid-1990s by Alistair Cockburn, an American computer scientist, known as one of the initiators of the agile movement in software development. (right figure)

The methodologies' name originates from the gemstone, crystal, matching it's geometrical shape of flat faces with specific, characteristic orientations with different views on the "underlying core" of principles and values representing techniques, tools, standards and roles.

After years from studies and interviews of teams dealing with software development, Cockburn's research showed that teams that did not follow formal methodologies delivered successful projects. This methods explain why, having as focus people, their interactions, community, skills, talents and communications with the belief that these are what have the first-order effect on performance.

There are different methods of this family, developed to suit teams of different sizes which need different strategies to solve their problems. The names of each methodology use colours to denote their "weight" based on project size and criticality (life, essential and discretionary money and comfort). The larger the project gets, the darker the colour, as can be seen in the figures right and below. The most common are (from smaller to larger):

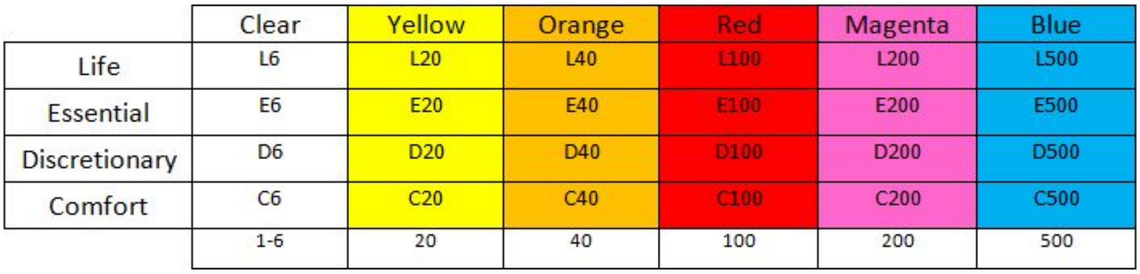


| | Clear | Yellow | Orange | Red | Maroon |
|---|---|---|---|---|---|
| Life (L) | L6 | L20 | L40 | L80 | L200 |
| Essential Money (E) | E6 | E20 | E40 | E80 | E200 |
| Discretionary Money (D) | D6 | D20 | D40 | D80 | D200 |
| Comfort (C) | C6 | C20 | C40 | C80 | C200 |
| | 1-6 | 7-20 | 21-40 | 41-80 | 81-200 |

Crystal Clear
Crystal Yellow
Crystal Orange

Crystal Orange Web
Crystal Red
Crystal Maroon

Crystal Diamond
Crystal Sapphire



| | Clear | Yellow | Orange | Red | Magenta | Blue |
|---|---|---|---|---|---|---|
| Life | L6 | L20 | L40 | L100 | L200 | L500 |
| Essential | E6 | E20 | E40 | E100 | E200 | E500 |
| Discretionary | D6 | D20 | D40 | D100 | D200 | D500 |
| Comfort | C6 | C20 | C40 | C100 | C200 | C500 |
| | 1-6 | 20 | 40 | 100 | 200 | 500 |

Cockburn redefines this different terms as:

- Methodology - set of elements (e.g. practices, tools)
- Techniques - skill areas (e.g. developing use cases)
- Policies - dictate organizational musts

This methods have seven properties in common between them:

### 1. Frequent Delivery

Regular releasing of iterations of the software program, an idea integrated directly from agile methodologies. The developers decide which features they must include in each release and this is what is developed and tested in that time. In this family of methods, the updates should be weekly or quarterly (at maximum, up to 4 months for large, highly critical projects) and can be more than one iteration in a release. Usually a collection of iterations are gathered and delivered in a single release.

### 2. Reflective Improvement

It is related to the above and involves taking a break from regular development to find better ways for processes. The regular iterations provide feedback on whether the process is working. It usually is encouraged to do a meeting every couple of weeks to discuss what can be modified in the project.

### 3. Close or Osmotic Communication

Osmotic communication stands for the team being together in a room and getting information to flow around it, instead of just communicating by email or other means. It signifies that questions that arise from the work can be rapidly answered reducing the errors that provide from different interpretations.

### 4. Personal Safety

The people working on the project must be able to trust each other and feel free to speak up about issues or whatever arises so that the last point works.

### 5. Focus

Focus refers to firstly focusing on an individual task in a project for enough time that progress will be made including the issues that could affect it and secondly, to the direction of which the project is heading. There are two rules for dealing with this issue: assigning a two-hour-free of interruptions period for each developer and each developer is assigned to a functionality for two days before being switched to another one.

### 6. Easy access to expert users

This involves the developers working with a person of expertise (that should be an actual/real-life user and not just a tester from the development team) in the project area so that the expert answers any questions.

### 7. Technical environment with automated tests, configuration management and frequent integration

There should be continuous integration and testing so that if any changes are made, then errors, breakages, etc can be spotted which can be done by checking-in code into a repository, helping in identifying the problem code and remove it by reverting back or updating with correct code.

The Crystal approach defines a number of roles:

Project Sponsor whose responsibility is to finance the project and deliver the mission statement;
Senior Designer/Programmer which maintains the team structure, implements methodology and designs the system;
Designer/Programmers (Business Class Designers, Programmers, Software Documenters and Unit Testers), responsible for creating screen drafts, design sketches and notes, common object models, source code, packaged system, migration code, and test cases;
Users which help with use case and screen drafts.

Also, there are a number of other roles such as Architect, Coordinator, Requirements Gatherer, Business Expert, Business Analyst/Designer, Project Manager, Design Mentor, Usage Expert, Lead Design Programmer, UI designer, Technical Facilitator and Technical Writer.

Strategies used:

- Exploratory 360°
- Early victory
- Walking skeleton
- Incremental
- Re-architecture
- Information radiators

Techniques:

- Methodology shaping
- Reflection workshop,
- Blitz planning,
- Delphi estimation using expertise ranking,
- Daily standup meetings,
- Essential interaction design,
- Process miniature,
- Side-by-side programming,
- Burn charts

# Spiral Methodology

This model was described for the first time by Barry Boehm (right figure) in his 1986 article, "The Spiral Model of Software Development and Enhancement". In 1988, Boehm published a similar article for a wider public. This articles present a diagram that was subsequently published in a lot of different discussions about this model. This first articles use the expression of “Process Model” in regards to this method as well as to incremental, waterfall, prototyping, and other approaches, characterizing, however, the spiral model's risk-driven blending of other process models' features.

The Spiral Lifecycle Model is similar to the Incremental Model with greater emphasis in risk analysis. There are 4 phases to this process:

1. **Identification**

   This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase, including Business (BRS) and System (SRS) Requisites Specifications. It also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

2. **Risk Analysis**

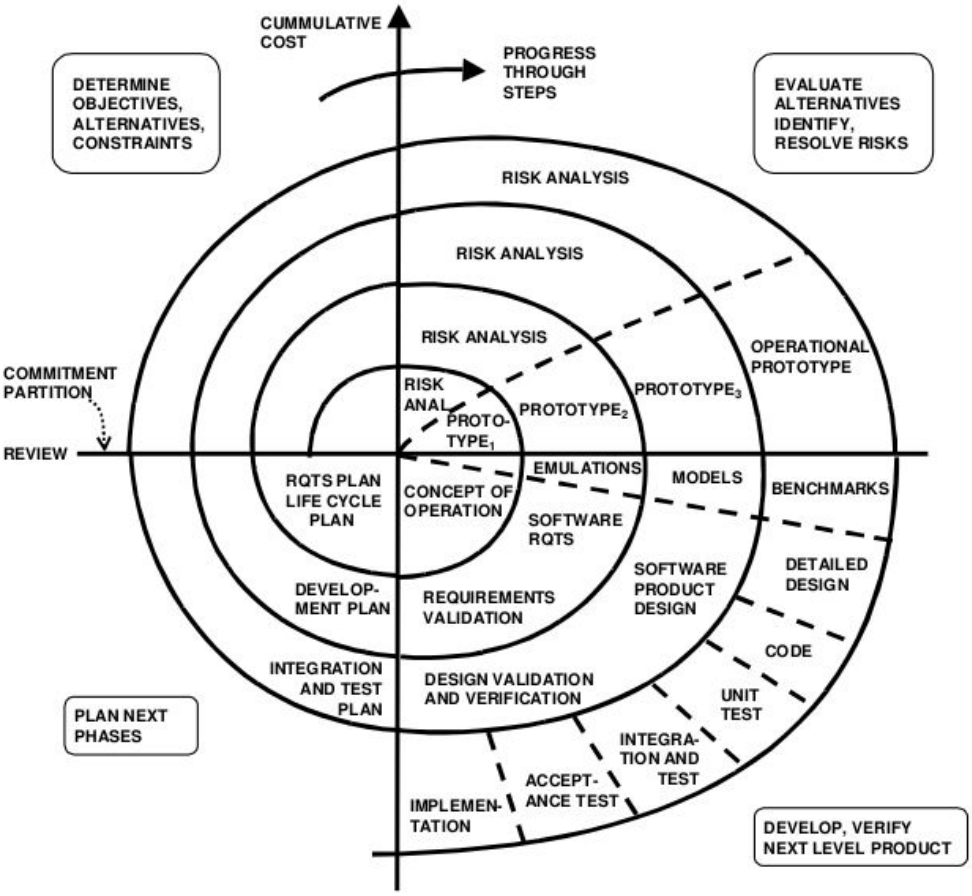
   Potential risks are identified and evaluated and the alternatives in question are also evaluated. The risks are recorded, assessed, and then reduced using prototypes, simulations, and analysis software. In this cycle, several prototypes exist as design templates or functional components.

3. **Development and Test**

   The prototypes are further expanded and functionalities are added. The actual code is written, tested, and migrated to a test environment several times until the software can be implemented in a productive environment.

## 4. Evaluation and Plan of Next Iteration

The next cycle is planned at the end of each cycle. If errors occur, solutions are looked for, and if an alternative is a better solution, it is preferred in the next cycle. Customers evaluate the software and provide their feedback and approval.



As can be seen in the figure above, the most important driving force of the spiral model is the risk analysis and assessment. Any risk that threatens the project is supposed to be identified from the beginning since the progress of the project is dependent of the possibilities of eliminating them. The project is considered successful only once there are no risks. The purpose of the cycle is to produce a continuously improving product as the software is constantly refined. The spiral model is incremental, but not necessarily iterative. Iterations occur only when risks, errors or conflicts threaten the project. Then the product has to go through one cycle again. There is no specific duration for each cycle as it depends on the anchor point milestones. As for team roles, there are no roles defined as this method os not considered a framework.

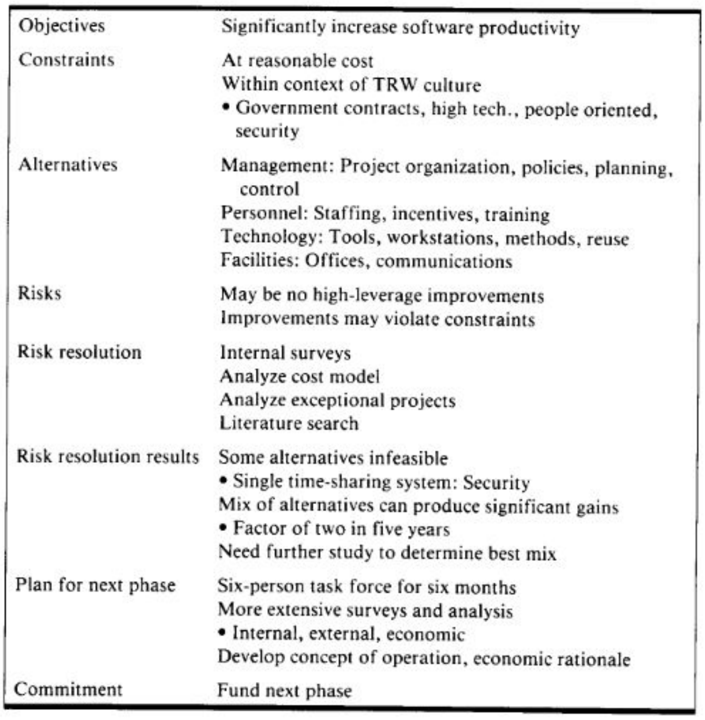
## Benefits and Disadvantages

- The spiral model is often used for larger, new technical environment projects that are subject to risks and need a great control of the budgets for clients and companies.
- Conflicts between the requirements for a software and its design are effectively avoided by the cyclic approach, since the requirements can be constantly checked and, if necessary, changed.
- Feedback can be obtained from users, developers, and clients at early project phases. However, this structure also necessitates management, which has the cycles of the product in view and can respond promptly to risks. The control of such projects is therefore relatively complex and also requires a good documentation so that all changes are recorded.
- Although the software is tested under various aspects during the development and testing cycle (unit, acceptance and integration test), it often happens that prototypes get transferred to the production system. There is therefore a risk that other errors and conceptual inconsistencies will be entered into the later end product.
- In places where decisions are made about the following cycles, there is a risk that loops will form and the project will take longer if wrong decisions are made. For this reason, the alternatives and their evaluation are important.

## Example of Utilization: The TRW Software Productivity System

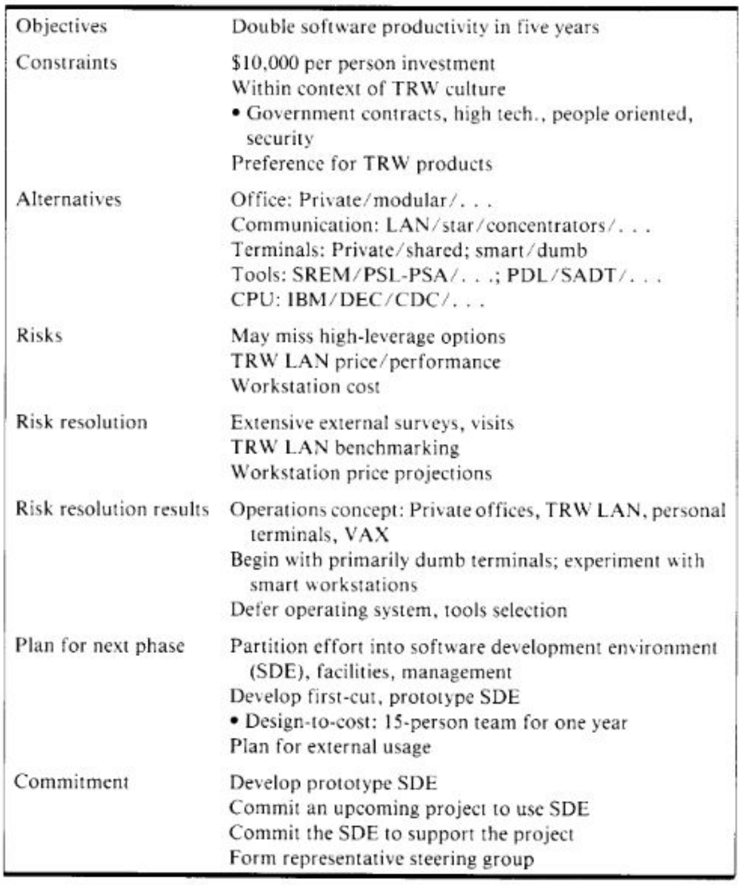
The TRW Software Productivity System (SPS) was an integrated software support environment based on the Unix operating system, a wide range of TRW software tools, and a wideband local network. The initial mission opportunity coincided with a corporate initiative to improve productivity in all appropriate corporate operations and an initial hypothesis that software engineering was an attractive area to investigate. This led to a small, extra “Round 0” circuit of the spiral to determine the feasibility of increasing software productivity at a reasonable corporate cost. (Very large or complex software projects will frequently precede the “concept of operation” round of the spiral with one on more smaller rounds to establish feasibility and to reduce the range of alternative solutions quickly and inexpensively.) The following three images document the first rounds of this system development using the Spiral Lifecycle Model.

**Table 1. Spiral model usage: TRW Software Productivity System, Round 0.**



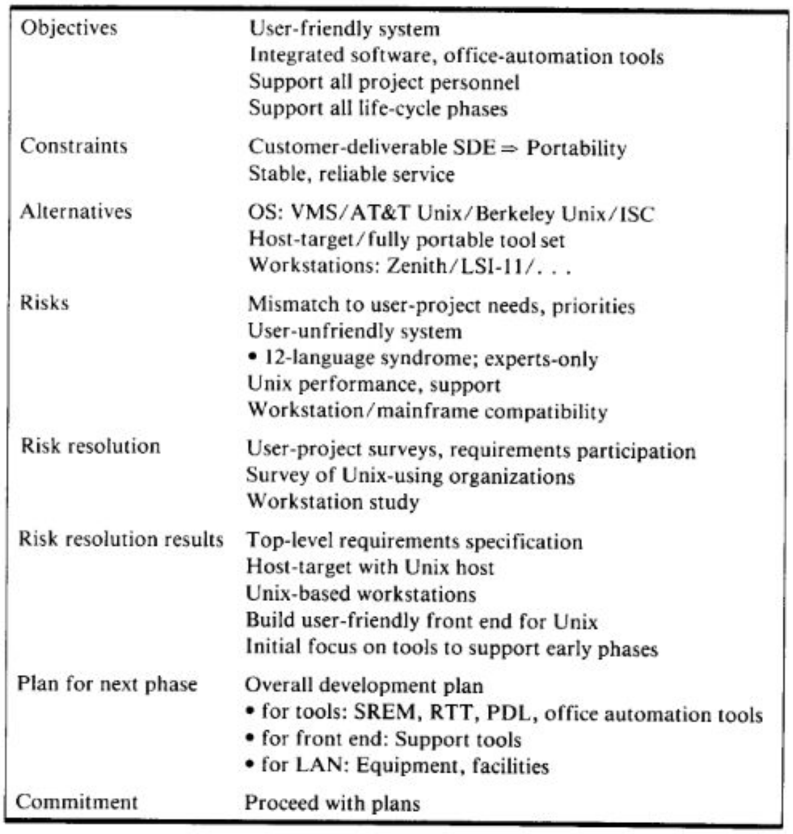
| | |
|---|---|
| Objectives | Significantly increase software productivity |
| Constraints | At reasonable cost<br>Within context of TRW culture<br>• Government contracts, high tech., people oriented, security |
| Alternatives | Management: Project organization, policies, planning, control<br>Personnel: Staffing, incentives, training<br>Technology: Tools, workstations, methods, reuse<br>Facilities: Offices, communications |
| Risks | May be no high-leverage improvements<br>Improvements may violate constraints |
| Risk resolution | Internal surveys<br>Analyze cost model<br>Analyze exceptional projects<br>Literature search |
| Risk resolution results | Some alternatives infeasible<br>• Single time-sharing system: Security<br>Mix of alternatives can produce significant gains<br>• Factor of two in five years<br>Need further study to determine best mix |
| Plan for next phase | Six-person task force for six months<br>More extensive surveys and analysis<br>• Internal, external, economic<br>Develop concept of operation, economic rationale |
| Commitment | Fund next phase |

**Table 2. Spiral model usage: TRW Software Productivity System, Round 1.**



| | |
|---|---|
| Objectives | Double software productivity in five years |
| Constraints | $10,000 per person investment<br>Within context of TRW culture<br>• Government contracts, high tech., people oriented, security<br>Preference for TRW products |
| Alternatives | Office: Private/modular/. . .<br>Communication: LAN/star/concentrators/. . .<br>Terminals: Private/shared; smart/dumb<br>Tools: SREM/PSL-PSA/. . .; PDL/SADT/. . .<br>CPU: IBM/DEC/CDC/. . . |
| Risks | May miss high-leverage options<br>TRW LAN price/performance<br>Workstation cost |
| Risk resolution | Extensive external surveys, visits<br>TRW LAN benchmarking<br>Workstation price projections |
| Risk resolution results | Operations concept: Private offices, TRW LAN, personal terminals, VAX<br>Begin with primarily dumb terminals; experiment with smart workstations<br>Defer operating system, tools selection |
| Plan for next phase | Partition effort into software development environment (SDE), facilities, management<br>Develop first-cut, prototype SDE<br>• Design-to-cost: 15-person team for one year<br>Plan for external usage |
| Commitment | Develop prototype SDE<br>Commit an upcoming project to use SDE<br>Commit the SDE to support the project<br>Form representative steering group |

**Table 3. Spiral model usage: TRW Software Productivity System, Round 2.**



| | |
|---|---|
| Objectives | User-friendly system<br>Integrated software, office-automation tools<br>Support all project personnel<br>Support all life-cycle phases |
| Constraints | Customer-deliverable SDE ⇒ Portability<br>Stable, reliable service |
| Alternatives | OS: VMS/AT&T Unix/Berkeley Unix/ISC<br>Host-target/fully portable tool set<br>Workstations: Zenith/LSI-11/. . . |
| Risks | Mismatch to user-project needs, priorities<br>User-unfriendly system<br>• 12-language syndrome; experts-only<br>Unix performance, support<br>Workstation/mainframe compatibility |
| Risk resolution | User-project surveys, requirements participation<br>Survey of Unix-using organizations<br>Workstation study |
| Risk resolution results | Top-level requirements specification<br>Host-target with Unix host<br>Unix-based workstations<br>Build user-friendly front end for Unix<br>Initial focus on tools to support early phases |
| Plan for next phase | Overall development plan<br>• for tools: SREM, RTT, PDL, office automation tools<br>• for front end: Support tools<br>• for LAN: Equipment, facilities |
| Commitment | Proceed with plans |

# Group:

- Nádia de Sousa Varela de Carvalho, up201208223
- Lívia Carolinne Trindade de Lima Araújo, up201702858
- António José Magalhães da Cruz Magalhães, up201404375

# References:

Crystal Methods

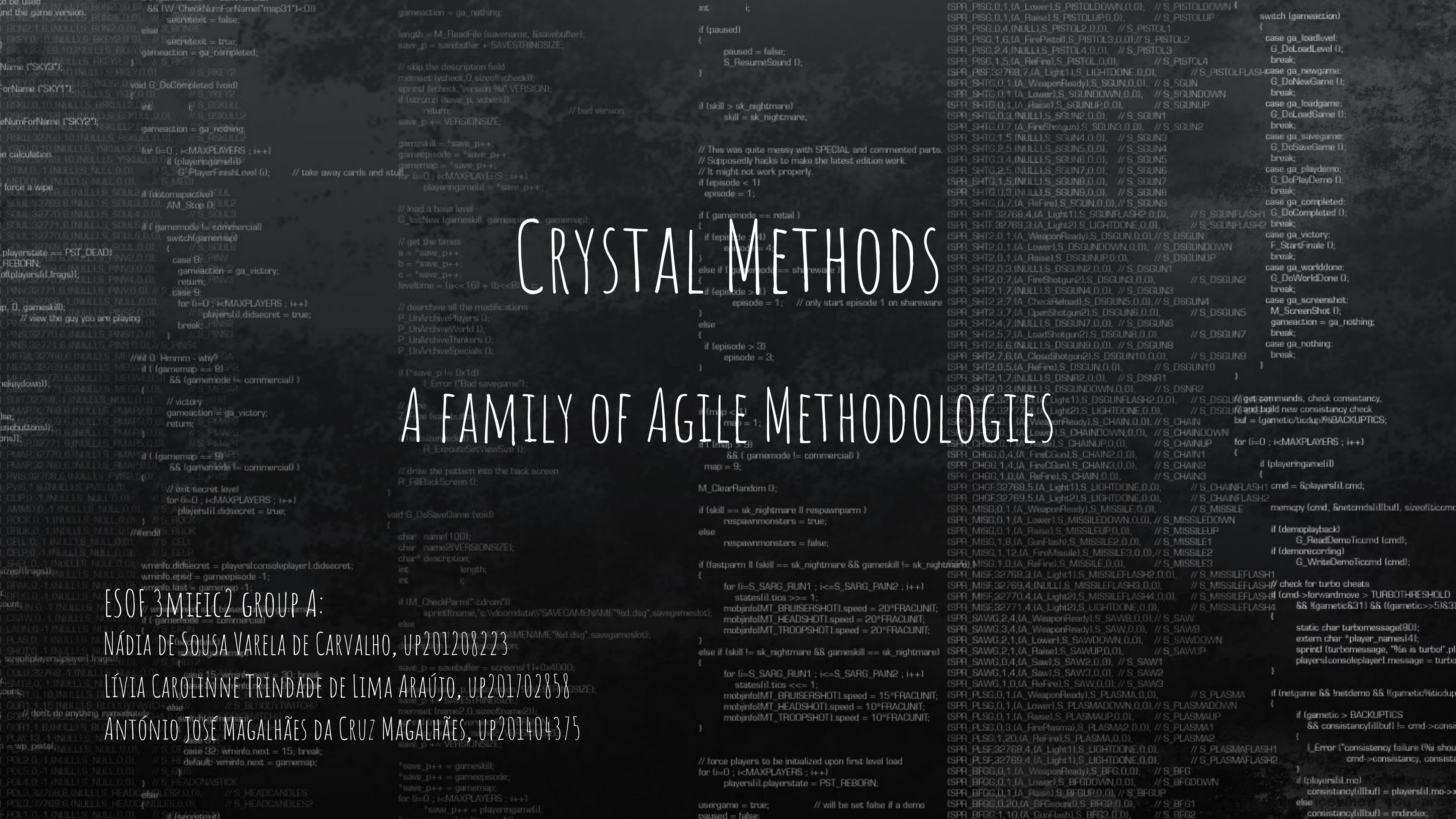
- itinfo.am
- Pratical Guide to 7 Agile Methods - part 2
- Alistair Cockburn views on Osmotic Communication

- Stepanek, G. Close/Osmotic Communication in Software Programming Secrets: Why Projects Fail
- Reijnen, C. Investigating The Crystal Clear Method
- Stephens, R. Beginning Software Engineering
- Flora, H. Chande, V. A Systematic Study on Agile Software Development Methodologies and Practices

Spiral Lifecycle Model

- Ryte
- What is Spiral model- advantages, disadvantages and when to use it?
- XbSoftware
- Software Development Life Cycle

- Boehm, B. A Spiral Model Of Software Development and Enhancement
- Boehm, B. Hansen, W. J. Spiral Development: Experience, Principles, and Refinements
- Boehm, B. Penedo, M. Memories of TRW's Software Productivity Project



# Crystal Methods

## A family of Agile Methodologies

ESOF 3MIEIC2 group A:

Nádia de Sousa Varela de Carvalho, up201208223

Lívia Carolinne Trindade de Lima Araújo, up201702858

António José Magalhães da Cruz Magalhães, up201404375

# History

- Developed by **Alistair Cockburn**
- Mid-1990s
  - 1992: Start
  - 1997: named “Crystal”
  - 2001: Agile Manifesto
- It was one of the founding ‘agile’ methodologies.

Years of studies and interviews showed that teams that did not follow formal methodologies delivered successful projects.

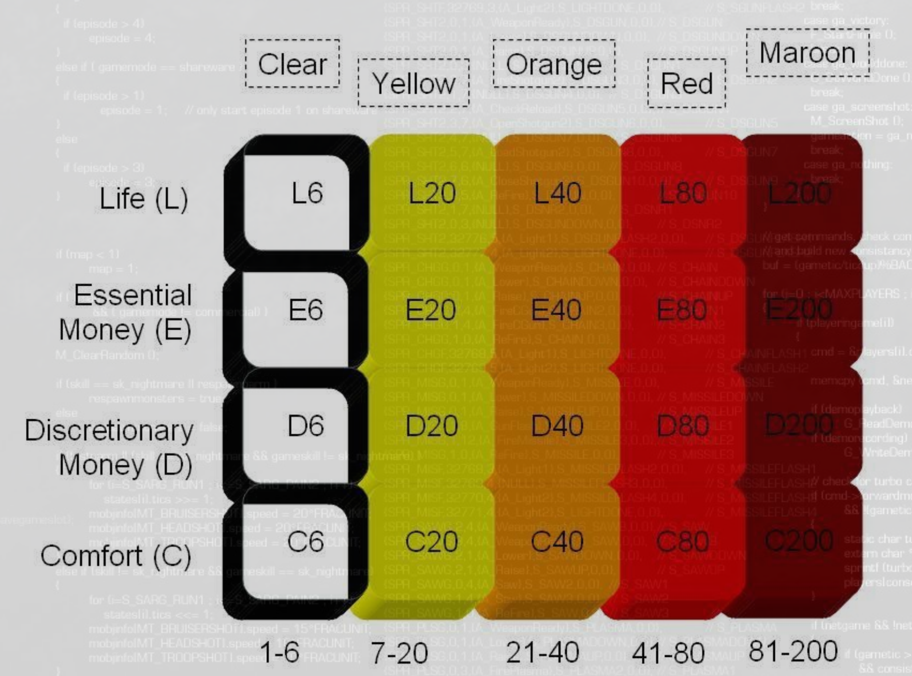
This methods explain why, having as focus:

- People
- Interactions
- Community
- Skills
- Talents
- Communication

# Family of methodologies

Organized by size and criticality:

- Crystal Clear
- Crystal Yellow
- Crystal Orange
- Crystal Orange Web
- Crystal Red
- Crystal Maroon
- Crystal Diamond
- Crystal Sapphire



| | Clear | Yellow | Orange | Red | Maroon |
|---|---|---|---|---|---|
| Life (L) | L6 | L20 | L40 | L80 | L200 |
| Essential Money (E) | E6 | E20 | E40 | E80 | E200 |
| Discretionary Money (D) | D6 | D20 | D40 | D80 | D200 |
| Comfort (C) | C6 | C20 | C40 | C80 | C200 |
| | 1-6 | 7-20 | 21-40 | 41-80 | 81-200 |

- Methodology - set of elements (e.g. practices, tools)

- Techniques - skill areas (e.g. developing use cases)

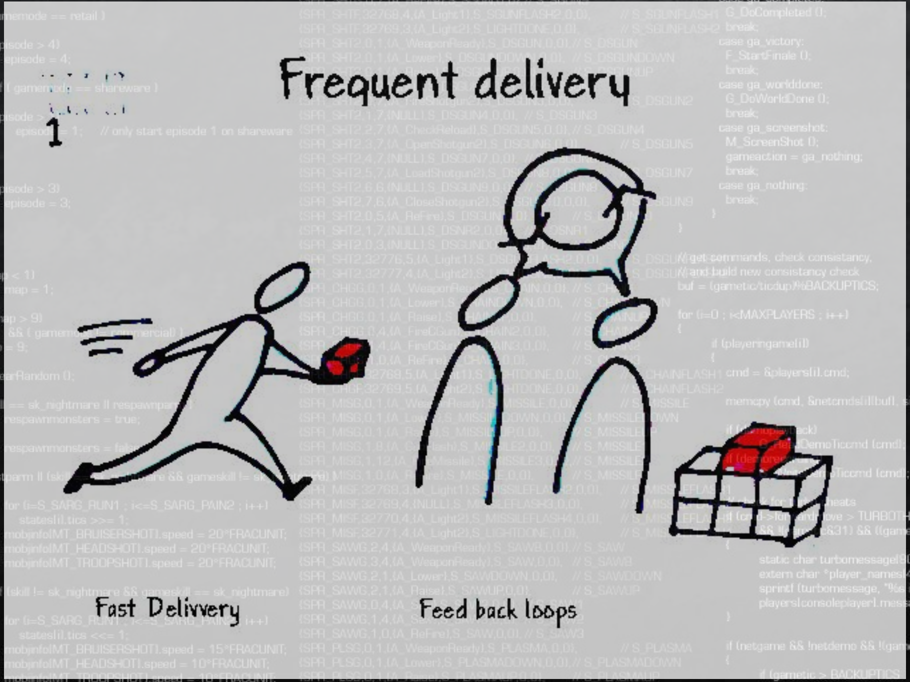
- Policies - dictate organizational musts

# Proprieties

- Frequent Delivery
- Reflective Improvement
- Close or Osmotic Communication
- Personal Safety
- Focus
- Easy access to expert users
- Technical environment with automated tests, configuration management and frequent integration

# FREQUENT DELIEVERY

- WEEKLY OR QUARTERLY
- MAXIMUM 4 MONTHS
- DECIDE FEATURES THEY MUST INCLUDE IN EACH RELEASE
- TEST AND USE OF FUNCTIONALITY AFTER EACH ITERATION



# Reflective Improvement

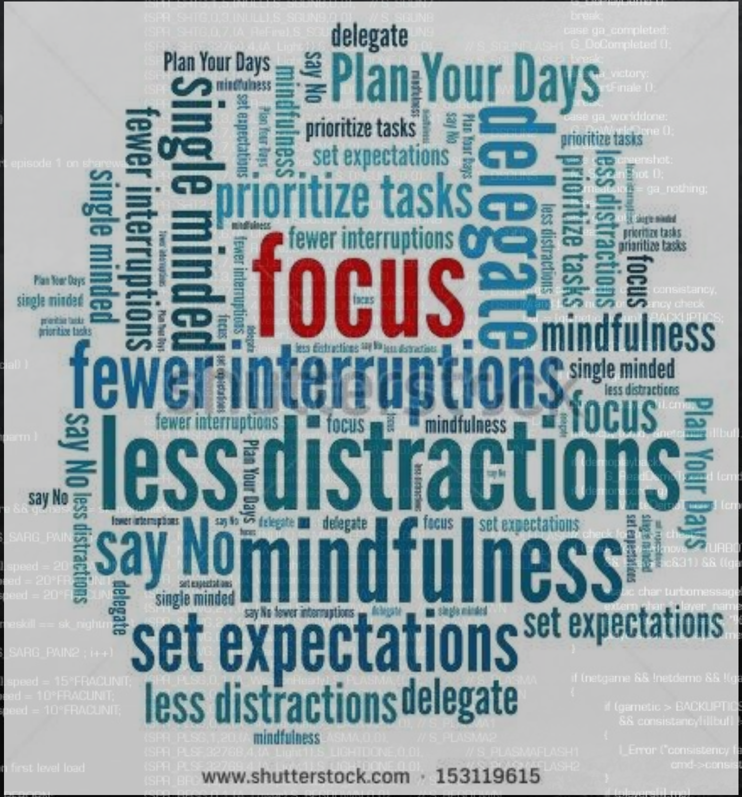
- Taking a break from regular development to find better ways for processes
- Meeting every couple of weeks to discuss what can be modified in the project

# Close or Osmotic Communication and Personal Safety

- team being together in a room and getting information to flow around it
- questions that arise from the work can be rapidly answered, reducing the risk of errors
- people working on the project must be able to trust each other and feel free to speak up about issues

# Focus

- Individual task
  - ➢ Two-hour-free of interruptions period for each developer
- Direction the project is heading
  - ➢ Each developer is assigned to a functionality for two days before being switched to another one



# Easy access to expert users

- Working with a person of expertise:
  - Actual/real-life user
- Expert answers any questions

# Technical environment with automated tests, configuration management and frequent integration

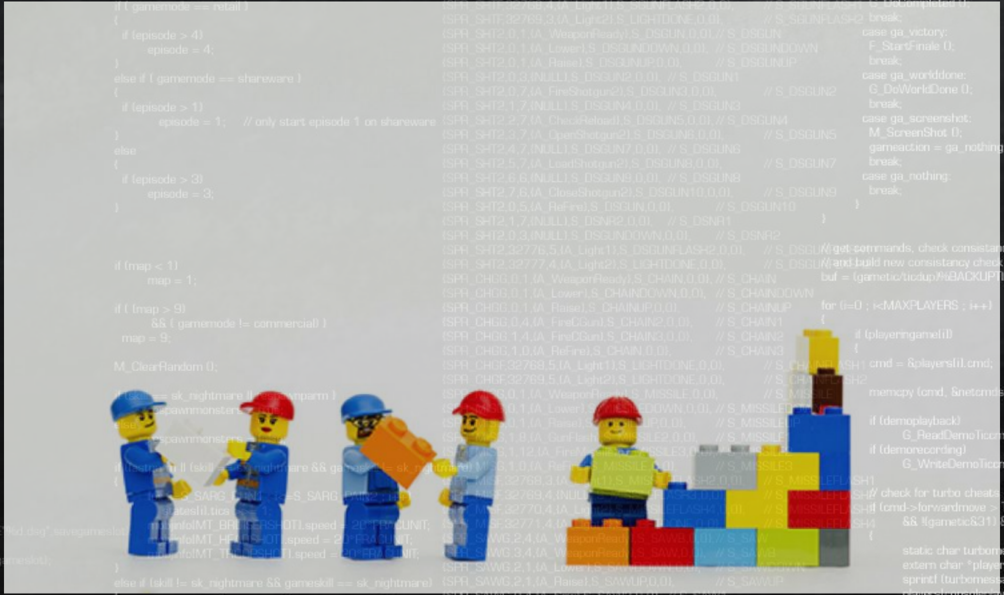
- Continuous integration and testing
- Errors, breakages, etc can be spotted
- Checking-in code into a repository
- Remove it by reverting back or updating with correct code

# Roles in the Team

- **Project Sponsor:** finance the project, deliver the mission statement
- **Senior Designer:** maintains the team structure, implements methodology and designs the system
- **Designers/Programmers:** creating screen drafts, design sketches and notes, common object models, source code, packaged system, migration code, and test cases
    - Business Class Designers
    - Unit Testers
    - Software "Documenters"
    - Programmers
- **Users:** help with use case and screen drafts

# Strategies

- Exploratory 360°
- Early victory
- Walking skeleton
- Incremental
- Re-architecture
- Information radiators



# Techniques

- Methodology shaping
- Reflection workshop
- Blitz planning
- Delphi estimation using expertise ranking
- Daily stand-up meetings
- Essential interaction design
- Process miniature
- Side-by-side programming
- Burn charts