

Crystal Methods and Spiral Lifecycle Model

An agile and a traditional software development methodologies

Crystal Methods

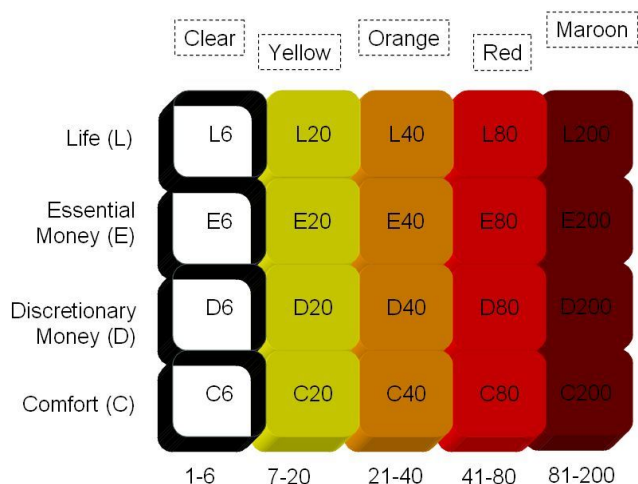
Crystal Methods are a family of methodologies developed in the mid-1990s by Alistair Cockburn, an American computer scientist, known as one of the initiators of the agile movement in software development. (right figure)



The methodologies' name originates from the gemstone, crystal, matching its 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):



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 these different terms as:

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

These 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

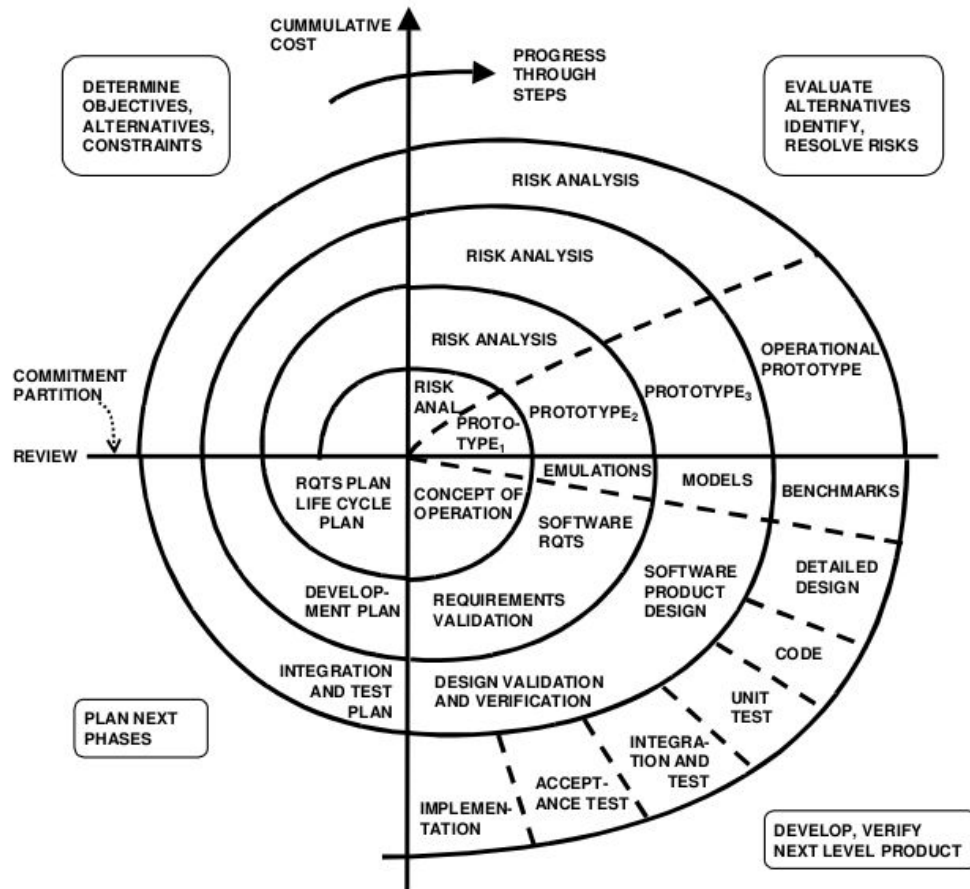
Potential risks are identified and evaluated and the alternatives in question are also evaluated. The risks are recorded, assessed, and then reduced using prototypes, simulations, and analysis software. In this cycle, several prototypes exist as design templates or functional components.

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 is 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 or 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 Within context of TRW culture • Government contracts, high tech., people oriented, security
Alternatives	Management: Project organization, policies, planning, control Personnel: Staffing, incentives, training Technology: Tools, workstations, methods, reuse Facilities: Offices, communications
Risks	May be no high-leverage improvements Improvements may violate constraints
Risk resolution	Internal surveys Analyze cost model Analyze exceptional projects Literature search
Risk resolution results	Some alternatives infeasible • Single time-sharing system: Security Mix of alternatives can produce significant gains • Factor of two in five years Need further study to determine best mix
Plan for next phase	Six-person task force for six months More extensive surveys and analysis • Internal, external, economic 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 Within context of TRW culture • Government contracts, high tech., people oriented, security Preference for TRW products
Alternatives	Office: Private/modular/. . . Communication: LAN/star/concentrators/. . . Terminals: Private/shared; smart/dumb Tools: SREM/PSL-PSA/. . .; PDL/SADT/. . . CPU: IBM/DEC/CDC/. . .
Risks	May miss high-leverage options TRW LAN price/performance Workstation cost
Risk resolution	Extensive external surveys, visits TRW LAN benchmarking Workstation price projections
Risk resolution results	Operations concept: Private offices, TRW LAN, personal terminals, VAX Begin with primarily dumb terminals; experiment with smart workstations Defer operating system, tools selection
Plan for next phase	Partition effort into software development environment (SDE), facilities, management Develop first-cut, prototype SDE • Design-to-cost: 15-person team for one year Plan for external usage
Commitment	Develop prototype SDE Commit an upcoming project to use SDE Commit the SDE to support the project Form representative steering group

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

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

Group:

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

References:

Crystal Methods

- itinfo.am
- [Practical 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](#)

[illegible]

```
// force players to be initialized upon first level load
for (i=0 ; i<MAXPLAYERS ; i++)
    players[i].playerstate = PST_REBORN;

usergame = true; // will be set false if a demo
paused = false;
```

```

( (SPR_PEGG.0.1.(A_Lower).S_PISTOLDOWN.0.0) // S_PISTOLDOWN
( (SPR_PEGG.0.1.(A_Raise).S_PISTOLUP.0.0) // S_PISTOLUP
( (SPR_PEGG.0.4.(NULL).S_PISTOL2.0.0) // S_PISTOL1
( (SPR_PEGG.1.6.(A_FirePistol).S_PISTOL3.0.0.(A_PISTOL2
( (SPR_PEGG.2.4.(NULL).S_PISTOL4.0.0) // S_PISTOL3
( (SPR_PEGG.1.5.(A_Refire).S_PISTOL4.0.0) // S_PISTOL4
( (SPR_PEGG32769.7.(A_Light1).S_LIGHTDONE.0.0) // S_PISTOLFLASH
( (SPR_SHTG.0.1.(A_WeaponReady).S_SGUN.0.0) // S_SGUN
( (SPR_SHTG.0.1.(A_SGUNDOWN).S_SGUNDOWN.0.0) // S_SGUNDOWN
( (SPR_SHTG.0.1.(A_Raise).S_SGUNUP.0.0) // S_SGUNUP
( (SPR_SHTG.0.3.(NULL).S_SGUN2.0.0) // S_SGUN1
( (SPR_SHTG.0.7.(A_FireShotgun).S_SGUN3.0.0) // S_SGUN2
( (SPR_SHTG.1.5.(NULL).S_SGUN4.0.0) // S_SGUN3
( (SPR_SHTG.2.5.(NULL).S_SGUN5.0.0) // S_SGUN4
( (SPR_SHTG.3.4.(NULL).S_SGUN6.0.0) // S_SGUN5
( (SPR_SHTG.2.5.(NULL).S_SGUN7.0.0) // S_SGUN6
( (SPR_SHTG.1.5.(NULL).S_SGUN8.0.0) // S_SGUN7
( (SPR_SHTG.0.3.(NULL).S_SGUN9.0.0) // S_SGUN8
( (SPR_SHTG.0.7.(A_Refire).S_SGUN.0.0) // S_SGUN9
( (SPR_SHTG2.0.3.(NULL).S_DSGUN1.0.0) // S_DSGUN1
( (SPR_SHTG.0.7.(A_FireShotgun2).S_DSGUN3.0.0) // S_DSGUN2
( (SPR_SHTG.1.7.(NULL).S_DSGUN4.0.0) // S_DSGUN3
( (SPR_SHTG2.2.(A_CloseShotgun).S_DSGUNE.0.0) // S_DSGUN4
( (SPR_SHTG2.3.7.(A_DownShotgun2).S_DSGUNE.0.0) // S_DSGUN5
( (SPR_SHTG2.4.7.(NULL).S_DSGUN7.0.0) // S_DSGUNG
( (SPR_SHTG.5.7.(A_LoadShotgun2).S_DSGUN8.0.0) // S_DSGUN7
( (SPR_SHTG2.6.6.(NULL).S_DSGUN9.0.0) // S_DSGUN8
( (SPR_SHTG2.7.6.(A_CloseShotgun2).S_DSGUN10.0.0) // S_DSGUN9
( (SPR_SHTG.0.5.(A_Refire).S_DSGUN.0.0) // S_DSGUN10
( (SPR_SHTG2.7.(NULL).S_DSGN2.0.0) // S_DSGN1
( (SPR_SHTG2.3.(NULL).S_DSGDOWN.0.0) // S_DSGNR2
( (SPR_SHTG32776.5.(A_Light1).S_LIGHTFLASH2.0.0) // S_DSGU1(Get commands, check consistency,
( (SPR_SHTG32777.4.(A_Light2).S_LIGHTDONE.0.0) // S_DSGU2(Get build now consistency check)
( (SPR_CHGC.0.1.(A_WeaponReady).S_CHAIN.0.0) // S_CHAIN
( (SPR_CHGC.0.1.(A_WeaponReady).S_CHAINDOWN.0.0) // S_CHAINDOWN
( (SPR_CHGC.0.1.(A_FireGun).S_CHAINUP.0.0) // S_CHAINUP
( (SPR_CHGC.0.1.(A_FireGun).S_CHAIN1.0.0) // S_CHAIN1
( (SPR_CHGC.0.1.(A_FireGun).S_CHAIN2.0.0) // S_CHAIN2
( (SPR_CHGC.0.1.(A_Refire).S_CHAIN3.0.0) // S_CHAIN3
( (SPR_CHGC32769.5.(A_Light1).S_LIGHTDONE.0.0) // S_CHAINFLASH1 cmd = $playersfil cmd;
( (SPR_CHGC32769.5.(A_Light2).S_LIGHTDONE.0.0) // S_CHAINFLASH2
( (SPR_CHGC.0.0.(NULL).S_MISSILE.0.0) // S_MISSILE
( (SPR_CHGC.0.0.(NULL).S_MISSILEDOWN.0.0) // S_MISSILEDOWN
( (SPR_CHGC.0.1.(A_WeaponReady).S_MISSILEUP.0.0) // S_MISSILEUP
( (SPR_CHGC.0.1.(A_WeaponReady).S_MISSILE2.0.0) // S_MISSILE2
( (SPR_CHGC.0.1.(A_WeaponReady).S_MISSILE3.0.0) // S_MISSILE3
( (SPR_MISSG.1.0.(A_Refire).S_MISSILE.0.0) // S_MISSILEFLASH1
( (SPR_MISSG32769.3.(A_Light1).S_MISSILEFLASH2.0.0) // S_MISSILEFLASH2
( (SPR_MISSG.0.0.(NULL).S_MISSILEFLASH.0.0) // S_MISSILEFLASH
( (SPR_MISSG.0.1.(A_Light1).S_LIGHTDONE.0.0) // S_MISSILEFLASH1
( (SPR_MISSG.0.1.(A_WeaponReady).S_BFG.0.0) // S_BFG
( (SPR_SAWG.0.1.(A_WeaponReady).S_SAW.0.0) // S_SAWB
( (SPR_SAWG.2.1.(A_Lower).S_SAWDOWN.0.0) // S_SAWDOWN
( (SPR_SAWG.2.1.(A_Raise).S_SAWUP.0.0) // S_SAWUP
( (SPR_SAWG.0.3.(A_FireSaw).S_SAW2.0.0) // S_SAW1
( (SPR_SAWG.1.4.(A_Saw).S_SAW2.0.0) // S_SAW2
( (SPR_SAWG.1.0.(A_Refire).S_SAW.0.0) // S_SAW3
( (SPR_SAWG.0.1.(A_WeaponReady).S_PLASMA.0.0) // S_PLASMA
( (SPR_PCSG.0.1.(A_Lower).S_PLASMADOWN.0.0) // S_PLASMADOWN
( (SPR_PCSG.0.1.(A_Raise).S_PLASMAUP.0.0) // S_PLASMAUP
( (SPR_PCSG.0.3.(A_FirePlasma).S_PLASMA2.0.0) // S_PLASMA1
( (SPR_PCSG.1.20.(A_Refire).S_PLASMA.0.0) // S_PLASMA2
( (SPR_PCSG32768.4.(A_Light1).S_LIGHTDONE.0.0) // S_PLASMAFLASH1
( (SPR_PCSG32768.4.(A_Light1).S_LIGHTDONE.0.0) // S_PLASMAFLASH2
( (SPR_PCSG.0.1.(A_WeaponReady).S_BFG.0.0) // S_BFG
( (SPR_PCSG.0.1.(A_Lower).S_BFGDOWN.0.0) // S_BFGDOWN
( (SPR_PCSG.0.1.(A_Raise).S_BFGUP.0.0) // S_BFGUP
( (SPR_PCSG.0.20.(A_FireGround).S_BFG2.0.0) // S_BFG1
( (SPR_PCSG.1.1.(A_GunFlash).S_BFG3.0.0) // S_BFG2

```



```

(SFR, PISG, 0, 1, (A, Lower), S_PISTOLDOWN, 0, 0), // S_PISTOLDOWN {
(SFR, PISG, 0, 1, (A, Raise), S_PISTOLDOWN, 0), // S_PISTOLUP
(SFR, PISG, 0, 1, (NULL), S_PISTOL, 0, 0), // S_PISTOL
(SFR, PISG, 1, 0, (A, FirePistol), S_PISTOL, 0, 0), // S_PISTOL2
(SFR, PISG, 2, 0, (A, FirePistol), S_PISTOL, 0, 0), // S_PISTOL3
(SFR, PISG, 1, 5, (A, Refire), S_PISTOL, 0, 0), // S_PISTOL4
(SFR, PISG, 3, 2, 7, (A, Light), S_LIGHTDONE, 0, 0), // S_PISTOLFLASH--case ga_newgame:
(SFR, SHTG, 0, 1, (A, WeaponReady), S_SGUN, 0, 0), // S_SGUN
(SFR, SHTG, 0, 1, (A, Lower), S_SGUNDOWN, 0, 0), // S_SGUNDOWN
(SFR, SHTG, 0, 1, (A, Raise), S_SGUNUP, 0, 0), // S_SGUNUP
(SFR, SHTG, 0, 0, (NULL), S_SGUN, 0, 0), // S_SGUN1
switch (gameaction)
{
case ga_loadlevel:
G_DeLoadLevel(1);
break;
case ga_newgame:
G_DeNewGame(1);
break;
case ga_loadgame:
G_DeLoadGame(1);
}

```

```

ISPR SHTG 0.7 (A, Finesthenig.S, SGUN3,0.0), //S, SGUN2
break;
ISPR SHTG 1.5 (NULL,S, SGUN4,0.0), //S, SGUN3
case ga_savegame:
G_DeSaveGame (0);
ISPR SHTG 2.5 (NULL,S, SGUN5,0.0), //S, SGUN4
break;
ISPR SHTG 3.4 (NULL,S, SGUN6,0.0), //S, SGUN5
case ga_playdemo:
G_DePlayDemo (0);
ISPR SHTG 2.5 (NULL,S, SGUN7,0.0), //S, SGUN6
break;

```

PEOPLE

INTERACTIONS

COMMUNITY

[illegible]

```

ISFH_CHRG 0.01 (A_Raise), S_LIGHTNO,0.0, // S_CHAIN0 for (i=0; i<MAXPLAYERS; i++)
ISFH_CHRG 0.01 (A_Chain), S_CHAIN2,0.0, // S_CHAIN1 {
ISFH_CHRG 0.01 (A_Chain), S_CHAIN3,0.0, // S_CHAIN2 { (playergame[i])
ISFH_CHRG 0.01 (A_Chain), S_CHAIN,0.0, // S_CHAIN3 {
ISFH_CHRG 0.01 (A_Light), S_LIGHTDONE,0.0, // S_CHAINFLASH1 cmd = &player[i].cmd;
ISFH_CHRG 32/69.5 (A_Light2), S_LIGHTDONE,0.0, // S_CHAINFLASH2

```

COMMUNICATION

```

(SFR, MSG, 32769, 0, Light1, MISSILEFLASH, SP2, 0, 0), // MISSILEFLASH
(SFR, MSG, 32769, 0, INULL, MISSILEFLASH, 0, 0), // MISSILEFLASH // check for turbo chests
(SFR, MSG, 32770, 0, Light2, MISSILEFLASH, 0, 0), // MISSILEFLASH // (cmd -> forwardmove) > TURBOTHRESHOLD
(SFR, MSG, 32771, 0, Light2, LIGHTDONE, 0, 0), // MISSILEFLASH
(SFR, SAWG, 3, 4, WeaponReady, SAW, 0, 0, 0), // SAW
(SFR, SAWG, 3, 4, WeaponReady, SAW, 0, 0, 0), // SAW
(SFR, SAWG, 3, 4, WeaponReady, SAW, 0, 0, 0), // SAW
static char turbomessage[60];
static char turbomessage[60];

```

```

ISF1_SAWG_2.1,(A_Lower,S_SAWDOWN_0.0), //S_SAWDOWN
ISF1_SAWG_2.1,(A_Rangel,S_SAWUP0.0), //S_SAWUP
ISF1_SAWG_0.4,(A_Saw1,S_SAW2_0.0), //S_SAW1
ISF1_SAWG_1.4,(A_Saw2,S_SAW3_0.0), //S_SAW2
ISF1_SAWG_1.4,(A_Ref1,S_SAW0.0), //S_SAW3
ISF1_PL3S_0.1,(A_WeaponReady,S_PLASMA_0.0), //S_PLASMA

```

```

//SPI_FLASH0.1(A_LowerP.S_FLASH0.DOWN.0.0) //S_FLASH0UP
//SPI_FLASH0.1(A_Ramse.S_FLASH0AUP.0.0) //S_FLASHAUP
//SPI_FLASH0.3(A_FireFlash.S_FLASH2A.0.0) //S_FLASHA1
//SPI_FLASH1.20(A_Retire.S_FLASHA.0.0) //S_FLASHA2
//SPI_FLASH32768(A_Light1.S_LIGHTDONE.0.0) //S_FLASHAFLASH1
//SPI_FLASH32768(A_Light1.S_LIGHTDONE.0.0) //S_FLASHAFLASH2

```

```
(ISF1, BF6G,0,1,1A, WeaponReady,1,S, BF6G,0,0), // S, BF6G
// (playerskill.mo)
(ISF1, BF6G,0,1,1A, Lower,1,S, BF6G,0,0), // S, BF6GDOWN
consistency(hillbuf) = playerskill.mo>
(ISF1, BF6G,0,1,1A, Raise,1,S, BF6UP,0,0), // S, BF6UP
else
(ISF1, BF6G,0,20,1A, BF6Gsound,1,S, BF6G,0,0), // S, BF6G1
(ISF1, BF6G,1,10,1A, GunFlash,1,S, BF6G,0,0), // S, BF6G2
consistency(hillbuf) = mIndex,
```

[illegible]

ORGANIZED BY SIZE AND CRITICALITY:

CRYSTAL CLEAR

```

MEGA_327696,NULLS_MEGA_417(gammamemle = B2
MEGA_327704,NULLS_MEGA_0.32(gammamemle = commercial)
mekydown))
MEGA_327716,NULLS_MEGA_0.01(gammamemle = commercial)

```

CRYSTAL ORANGE

CRYSTAL ORANGE WEB

```
char name[100];
char name2[VERSIONSIZE];
char* description;
int length;
```

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


```

ISPR_PISG.0,1,(A_Lower),S_PISTOLDOWN.0,0, //S_PISTOLDOWN {
ISPR_PISG.0,1,(A_Raise),S_PISTOLUP.0,0, //S_PISTOLUP { switch (gameaction) {
ISPR_PISG.0,4,(NULL),S_PISTOL2.0,0, //S_PISTOL1 { case ga_LoadLevel:
ISPR_PISG.1,6,(FirePistol),S_PISTOL3.0,0,1/S_PISTOL2 { case ga_DeLoadLevel:
ISPR_PISG.2,4,(NULL),S_PISTOL4.0,0, //S_PISTOL3 { break;
ISPR_PISG.1,5,(A_Retreat),S_PISTOL.0,0, //S_PISTOL4 { case ga_NewGame:
ISPR_PISG32768,7,(A_Light),S_LIGHTDONE.0,0, //S_PISTOLFLASH: case ga_DeNewGame:
ISPR_SHTC.0,1,(A_WeaponReady),S_SGUN.0,0, //S_SGUN { break;
ISPR_SHTC.0,1,(A_Lower),S_SGUNDOWN.0,0, //S_SGUNDOWN { break;
ISPR_SHTC.0,1,(A_Raise),S_SGUNUP.0,0, //S_SGUNUP { case ga_LoadGame:
ISPR_SHTC.0,3,(NULL),S_SGUN2.0,0, //S_SGUN1 { case ga_DeLoadGame:

```

```

(IPSP: SHTG,0.7,IA,ReFeed,SOUN2,0.0) // S_SOUN2
break;
case g: saveGame:
(IPSP: SHTG,1.5,INULL,SOUN4,0.0) // S_SOUN3
break;
case g: DeSaveGame:
(IPSP: SHTG,2.5,INULL,S_SOUN5,0.0) // S_SOUN4
break;
case g: playDemo:
(IPSP: SHTG,3.4,INULL,S_SOUN6,0.0) // S_SOUN5
break;
case g: playDemo:
(IPSP: SHTG,2.5,INULL,S_SOUN7,0.0) // S_SOUN6
break;
case g: DePlayDemo:
(IPSP: SHTG,1.5,INULL,S_SOUN8,0.0) // S_SOUN7
break;
case g: playDemo:
(IPSP: SHTG,0.9,INULL,S_SOUN9,0.0) // S_SOUN8
break;
case g: completed:
(IPSP: SHTG,0.7,IA,ReFeed,SOUN,0.0) // S_SOUN9
break;
case g: DeCompleted:
(IPSP: SHTG,27.68,4,IA,Light1,S_SOUNFLASH2,0.0) // S_SOUNFLASH2
break;
case g: DeCompleted:
(IPSP: SHTG,27.68,3,IA,Light2,S_SOUNFLASH2,0.0) // S_SOUNFLASH2
break;
case g: victory:
(IPSP: SHTG,0.7,IA,WinCondition,S_SOUNJUDG,0.0) // S_SOUNJUDG
break;

```

IPSPR	SH2T.0.1.0A_Lower1.DSGUNDOWN.0.01	// S_DSGUNDOWN	F_StartFrame 0;
IPSPR	SH2T.0.1.0A_Raise1.DSGUNUP.0.01	// S_DSGUNUP	break;
IPSPR	SH2T.2.0.0A_Raise1.DSGUN2.0.01	// S_DSGUN1	case ga_worldDone:
IPSPR	SH2T.2.0.0A_Raise1.DSGUN3.0.01	// S_DSGUN2	G_DeWorldDone 0;
IPSPR	SH2T.2.0.0A_Raise1.DSGUN4.0.01	// S_DSGUN3	break;
IPSPR	SH2T.2.2.7A_ChackReload.DSGUNS.0.01	// S_DSGUN4	case ga_screenShot:
IPSPR	SH2T.3.7.0A_OpenShotgun2.DSGUN6.0.01	// S_DSGUN5	G_ScreenShot 0;
IPSPR	SH2T.4.2.7A_Inull.DSGUN7.0.01	// S_DSGUN6	gameAction = ga_nothing;
IPSPR	SH2T.5.7.0A_LoadShotgun2.DSGUN8.0.01	// S_DSGUN7	break;
IPSPR	SH2T.6.6.INULL.DSGUN9.0.01	// S_DSGUN8	case ga_nothing:
IPSPR	SH2T.7.6.0A_CloseShotgun2.DSGUN10.0.01	// S_DSGUN9	break;

```

ISPR_SHT2.0.5.A.Refine().DSGUN0.0.0.1. //S.DSGUN10
ISPR_SHT2.1.7.(NULL).DSNR2.0.0.1. //S.DSNR1
ISPR_SHT2.0.3.(NULL).DSGUNDOWN.0.0.1. //S.DSGNR2
ISPR_CHG6.0.1.4.(NULL).Light11.S.DSGUNINFLASH2.0.0.1. //S.DSGUN10
ISPR_CHG6.0.1.4.(NULL).Light2.S.LIGHTDOWN.0.0.1. //S.DSGUN10
ISPR_CHG6.0.1.4.(NULL).Refine().S.CHAIN.0.0.1. //S.CHAIN
ISPR_CHG6.0.1.4.(NULL).CHANDOWN.0.0.1. //S.CHANDOWN
ISPR_CHG6.0.1.4.(NULL).S.CHAINUP.0.0.1. //S.CHAINUP
ISPR_CHG6.0.4.4.A.FineCgun.S.CHAIN2.0.0.1. //S.CHAIN1
ISPR_CHG6.1.4.4.A.FineCgun.S.CHAIN3.0.0.1. //S.CHAIN2
ISPR_CHG6.1.0.4.A.Refine().S.CHAIN.0.0.1. //S.CHAIN3

```

```

ISPR CHF32769.5(A, Light1), S, MISS1D0NE,0,0, // S_CHAINFLASH1 cmd = kPlayer101.cmd;
ISPR CHF32769.5(A, Light2), S, LIGHTD0NE,0,0, // S_CHAINFLASH2
ISPR MISS0.0(A, WeeperMissile), S, MISS1LE,0,0, // S_MISSILE memory (cmd, SnetcmdFlash1, level1)
ISPR MISS0.0(A, Tower), S, MISS1LEDOVA,0,0, // S_MISSILE_DVA
ISPR MISS0.0(A, Reave), S, MISS1LE,0,0, // S_MISSILEUP if (demoplayback)
ISPR MISS.1,B(A, GunFlash), S, MISS1LE2,0,0, // S_MISSILE1 G_ReadDemoCmd(cmd)
ISPR MISS.1,12(A, FirstMissile), S, MISS1LE3,0,0, // S_MISSILE2 if (demorecording)
nightmare ISPR MISS.0(A, ReFin), S, MISS1LE,0,0, // S_MISSILE3 G_WriteDemoCmd(cmd);
ISPR CHF32769.3(A, Light1), S, MISS1LEFLASH2,0,0, // S_MISSILEFLASH1
ISPR CHF32769.4,INULL, S, MISS1LEFLASH3,0,0, // S_MISSILEFLASH2 check for turbo cheats
ISPR CHF32770.0(A, Light2), S, MISS1LEFLASH4,0,0, // S_MISSILEFLASH3 if (isTurbo == TURBO_DISABLED)
ISPR CHF32770.0(A, Light3), S, MISS1LEFLASH5,0,0, // S_MISSILEFLASH4 if (isTurbo == TURBO_DISABLED)

```

```

// S_S_MISLEFTDASH // $gamedata/$GAMEDATA_FOLDER
(SPR_SAWG_2.4.IA.WeaponReady)S_SAWG_0.0.I // S_SAW
static char turbotmessage[90];
(SPR_SAWG_2.1.IA.Lower)S_SAWDOWN_0.0.I // S_SAWDOWN
extern char "player_names[4];
(SPR_SAWG_2.1.IA.Raise)S_SAWUP_0.0.I // S_SAWUP
sprintf(turbotmessage,"%s is turbo");
(SPR_SAWG_0.4.IA.Saw)S_SAW2_0.0.I // S_SAW1
playerconsoleplayer.message = turbo;
(SPR_SAWG_1.4.IA.Saw)S_SAW3_0.0.I // S_SAW2
}
(SPR_SAWG_1.0.IA.Refine)S_SAW_0.0.I // S_SAW3
}
(NETMSG_PSG_0.1.IA.WeaponReady)S_PLASMA_0.0.I // S_PLASMA
if(netgame && !netdemo && !gamedata$hidemo)
(NETMSG_PSG_0.1.A.Lower)S_PLASMDOWN_0.0.I // S_PLASMDOWN
$gamedata$BACKPACKETS
(NETMSG_PSG_0.1.IA.Raise)S_PLASMA_0.0.I // S_PLASMA
88 consistofall(Life,cmd>=0);

```

```

(SPR FL53.1 20.0A Refins PLASMA.0.0.0. // $ PLASMA2
(SPR FL53.22769.0.1A Light1.S LIGHTDOWN.0.0. // $ PLASMAFLASH1
(SPR FL53.22769.0.1A Light1.S LIGHTDOWN.0.0. // $ PLASMAFLASH2
(SPR BFG6.0.1.1A WeaponReady.S BFG.0.0.1. // $ BFG
(SPR BFG6.0.1.1A Lower.S BFGDOWN.0.0. // $ BFGDOWN
(SPR BFG6.0.1.1A Pump.S BFGUP.0.0.1. // $ BFGUP
(SPR BFG6.0.20.1A BFGSound.S BFG2.0.0. // $ BFG1
(SPR BFG6.1.10.1A GunFlash.S BFG3.0.0. // $ BFG2

```



```
switch (gameaction)
{
    case ga_loadlevel:
        G_DoLoadLevel ();
        break;
    case ga_newgame:
        G_DoNewGame ();
        break;
    case ga_loadgame:
        G_DoLoadGame ();
        break;
}
```

```
G_DeSaveDemo 0;
break;
case ga_playdemo:
G_DoPlayDemo 0;
break;
case ga_completed:
G_DeCompleted 0;
break;
case ga_victory:
F_Start finale 0;
break;
case ga_worldone:
G_DoWorldDone 0;
break;
case ga_screenshot:
M_ScreenShot 0;
gameaction = ga_nothing;
break;
case ga_nothing:
break;
}

// get semmands, check consistency,
// build new consistancy check
buf = (gametic/ticmap/%$BACKUPTS);
for i=0 ; i<$MAXPLAYERS ; ++i)
{
    # playerengame(i)
    {
        $FLASH1 cmd = %player%i.cmd;
        $FLASH2 memary tcmd, &netsent(i).buff, sizeof(ticom
            if ($?check DemoTcmd ccmd)
                $ticcmd ccmd;
            else {
                ticmap %i>%tmaps
                $S AS (>$31) && (%netics)>>$I83;
                static char turbomessage[90];
                extern char "player_names[4];
                sprintf(turbomessage,"%is turbo! pl
                    play%lconsoleplayr,message=turbo
            }
            if (!netgame && !nstdemo && $(gametic/ticdup
                # gameover - BACKUPTICS
                && consistency(i).buff != cmd->consist
            )Error "consistency failure (%d should ==> consistency, consiste
            }
            # player%i.mv)
            consistency(i).buff = players[i].mo->
        else
```

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



```

memory(save, p, description, SAVESTRINGSIZE);
save_p++=SAVESTRINGSIZE;
moment = fromm2(0, sizeof(mom2));
sprintf(mom2, "version %2d\n");
memory(save, p, mom2, VERSIONSIZE);
save_p+=VERSIONSIZE;

*save_p++ = gameskill;
*save_p++ = gamespeed;
*save_p++ = gamemap;
for (i=0; i<MAXPLAYERS; i++)
    *save_p++ = playeringame[i];

// force players to be initialized upon first level load
for (i=0; i<MAXPLAYERS; i++)
    players[i].playerstate = PST_REBORN;

username = true; // will be set false if a demo
paused = false;

```

```
{SPA_BFGG,0}
{SPA_BFGG,0}
{SPA_BFGG,1}
```

0.20, (A_BFGson
1, 10, (A_GunFla

```

und).S_BFG2,0
sh).S_BFG3,0

```

consistency will

```
buff = mdindex,
```


TEAM 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



[illegible]

FOCUS

The image is a complex collage. On the left, a vertical strip of text contains the words 'FOCUS', 'INTERRUPTIONS', 'PER', 'HEADING', 'NED TO A', 'DAYS BEFORE', and 'THER ONE' in a stylized, hand-drawn font. The main background is a collage of two images. The top image is a snippet of Lua code from a game, showing various game mechanics and player actions. The bottom image is a word cloud centered around the word 'focus', with other related terms like 'mindfulness', 'prioritize tasks', 'delegate', 'single minded', 'fewer interruptions', and 'less distractions' prominently displayed. The word cloud is set against a dark, textured background.

PROJECT SPONSOR: FINANCE THE PROJECT, DELIVER THE MISSION STATEMENT.
SENIOR DESIGNER: MAINTAINS THE TEAM STRUCTURE, IMPLEMENTS METHODOLOGY AND DESIGNS THE

DESIGNERS/PROGRAMMERS: CREATING SCREEN DRAFTS, DESIGN SKETCHES AND NOTES, COMMON OBJECT MODELS, SOURCE CODE, PACKAGED SYSTEM, MIGRATION CODE, AND TEST CASES

BUSINESS CLASS DESIGNERS

Software “DOCUMENTERS”

UNIT TESTERS PROGRAMMERS

```
if ( gamemode == commercial )
{
    spawn( name,SAVEGAME_NAME ".bak", savegame.slot );
}
else
{
    spawn( name,SAVEGAME_NAME ".bak", savegame.slot );
}
```

```
switch (gameaction)
{
    case ga_loadlevel:
        G_DoLoadLevel ();
        break;
    case ga_newgame:
        G_DoNewGame ();
        break;
    case ga_loadgame:
        G_DoLoadGame ();
        break;
}
```

```

// SAVING & LOADING
void SaveGame() {
    FILE *f = fopen("save.dat", "w");
    if (!f) return;
    fwrite(&game, sizeof(Game), 1, f);
    fclose(f);
}

void LoadGame() {
    FILE *f = fopen("save.dat", "r");
    if (!f) return;
    fread(&game, sizeof(Game), 1, f);
    fclose(f);
}

// GAME LOOP
int main() {
    srand((unsigned int)time(0));
    Game game;
    game.Init();
    while (true) {
        game.Run();
        if (game.IsGameOver()) break;
    }
    return 0;
}

```

```
case ga_savegame:
    G_DoSaveGame ();
    break;
case ga_playdemo:
    G_DoPlayDemo ();
    break;
case ga_completed:
    G_DoCompleted ();
    break;
case ga_victory:
    G_DoVictory ();
    break;
```

```

SH1  no = &playernr.ch
SH2  memcpy (cmd, &netc
      if (demoplayback)
          G_ReadDemoT
      if (demorecording)
          G_WriteDemo
FD SH1
S-# check for turbo ch
S-# (cmd->forward) mov
S-#  && lgametic&

```

```
static char tur[100];
extern char *p;
printf("turbo\n");
player[0] = 0;

}

if (netgame && !netdo)
{
    if (gametic > B)
        SS consista
```

```
FLASH1      I_Error ("con  
FLASH2      cr  
}  
if (players[il.m  
consistency  
else  
consistency
```