**Characteristics of the Settlers**

**Characteristics of the hardy pioneer millionaires – on TV (Circa 70’s):**

1. Willing to live in isolation to get the job/game completed (only home in their Sierra Nevada mountain valley)
2. Work long hours, 24 to 36 at a stretch
3. Work odd hours, start at 2 A.M. if the spirit so moved (come and go as they please)
4. Only phone, the private booth installed outside their home / office

**Seen as necessary by management to:**

1. Allow for innovation
2. Flexibility seen as key to attract the gurus
3. Accept that as a group, production may be erratic

**As the organization became more dependent on software, there has been an increasing desire to ensure:**

1. Reliable production schedules
2. Controllable production schedules that can be predicted
3. Reliable maintenance
4. Reduced, predictable error rates
5. Eliminate dependence on only one individual who knows the system

**Attempts to reduce dependence on exceptional individual effort, and produce predictable teams include:**

1. Traditional Teams
2. Egoless Teams
3. Group Reviews
4. walk through
5. inspections
6. Check List
7. Peer Review / Peer Rating
8. Group Coding, Testing and Debugging
9. RAD (Rapid Application Development)
10. JAD (Joint Application Development)
11. Tiger Teams
12. Agile

The best historical source is probably: “Software Psychology, Human Factors in Computers and Information Systems,” by Ben Shneiderman, Winthrop

**Individual Personality Traits of the Settlers**

Survey by Couger and Zawacki (1978)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Social Need** | **Growth Need** | **Motivation Potential** |
| **Data Processing Professionals** |  |  |  |
| **Sales** |  |  |  |
| **Other Professionals** (doctors, lawyers, etceteras) |  |  |  |
| **Service** (motels, food) |  |  |  |
| **Managerial** |  |  |  |
| **Clerical** |  |  |  |
| **Structural Workers** |  |  |  |

Computing professionals may be the most desirable group to manage in the workforce.

**Individual Personality Traits of the Settlers**

Survey by Couger and Zawacki (1978)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Social Need** | **Growth Need** | **Motivation Potential** |
| **Data Processing Professionals** | 4.19 | 6.02 | 157.5 |
| **Sales** | 5.41 | 5.70 | 146.0 |
| **Other Professionals** (doctors, lawyers, etcetera) | 5.48 | 5.59 | 153.7 |
| **Service** (motels, food) | 5.45 | 5.38 | 151.7 |
| **Managerial** | 5.65 | 5.30 | 155.9 |
| **Clerical** | 5.21 | 4.95 | 105.9 |
| **Structural Workers** | 5.14 | 4.54 | 140.4 |

Computing professionals may be the most desirable group to manage in the workforce.

**Conventional Team**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  | Senior Project Leader | | |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  | Project Leader | | |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Senior Programmer. Team w/ 3 to 10 other Programmers | | |  | Senior Programmer. Team w/ 3 to 10 other Programmers | | |
|  |  |  |  |  |  |  |

* 1. A Senior Programmer directs about 3 to 10 junior members of the team. The Senior Programmer determines work products (design), assigns work, collects, assimilates, and evaluates the work products.
  2. Work is done individually and competition between team members is seen as a stimulus to higher productivity.

Traditional approach is appealing because (software psychology):

1. Responsibility / authority lines are clear.
2. Reflects the way many homes, work environments, political, and many social systems are organized.
3. The system is easy to understand.
4. Competence is rewarded by promotion / task.
5. Individuals treated as professionals, typically nominal supervision.

Note that if the Senior Programmer praises an individual in the presence of the group for their contributions, the individual tends to work even harder and seek more responsibility.

**Critics Review**

1. Programming is not like fields such as marketing where each sales person accomplishes similar task but interaction is low.
2. Designers and programmers are highly specialize individuals working on highly interrelated parts of the same project.
3. Design and implementation require special skills. An individual cannot be expected to have the entire skill set on large projects.
4. The errors or delay caused by one individual can cause very adverse consequences for others but not always in an obvious manner.
5. The most competent implementers are promoted to managerial positions just as they reach the peak of the technical and application knowledge – business loses expertise! This occasionally reflects the Peter Principle (promote till incompetent).
6. Many companies advertise dual tracks: technical and management, but do not in fact reward both tracks equally.
7. The chance of technical personnel succeeding as managers is actually quite high if:
8. They are technically competent and command respect for their accomplishments.
9. Make reasonable demands of those they manage.
10. Reward good work as opposed to taking the credit.
11. Encourage a balance between competition and cooperation.

Managers seldom succeed in technical positions!

8) Managers may be appointed from other areas (military, government - plug compatible GS Level) and not be technically competent or competent in the area they are to manage. It is harder for them to command respect and more difficult to manage personnel and work products effectively.

**Egoless Teams**

(Gerald Weinberg 1971)

1. Egoless programming/development is a state of mind in which developers can separate themselves from the product. The separation allows them to accept suggestions without feeling criticism.
2. Developers are able to seek advice/help without fear of showing incompetence. Cooperation as opposed to competition is encouraged.
3. No one is formally placed in charge. The group determines its leadership.
4. An egalitarian community atmosphere replaces the harsh authoritarian lines of the conventional team.
5. Emphasis is placed on cooperation as opposed to competition. If a member of the teams experiences difficulty, others willing contribute their efforts.
6. Project definition, design, goals, and assignments are made by the entire team.
7. Team has the power to hire and fire. It must also be responsible for discipline if required. The group determines merit.

**Critics:**

1. No authority figure to reward or punish.
2. With no one in charge, the group is like a ship without a rudder. It may be difficult to maintain group focus, especially for long periods.

**Supporters:**

Egoless teams tend to produce high quality work, exhibit high morale, low turnover, and very loyal employees. Teams meet a social need leading to high productivity. Individual professional worth is emphasized!

**Chief Programmer Teams**

**Introduced in the early 70’s as a response to faults in traditional teams.**

1. Recognition that system specification, design, and implementations were increasingly complex tasks requiring specialized skill sets.
2. Desire to re-introduce the most experienced personnel back into the systems implementation track.
3. Desire to have junior personnel learn technical and corporate culture from the most qualified individuals as opposed to each other. Analogy of medical residents learning from the chief surgeon.

Results of IBM experiment to create a Morgue File System for the New York Times (results published in the early to mid-70’s).

**Team Members:**

1. **Chief Programmer**: Primary systems architect. Recognition that the ability to design is not evenly distributed in the market place. The chief programmer was responsible for the overall design, assignment of sub task to junior personnel, integration of junior personnel work into the product, directs testing, and quality control. Primary management responsibility for project including staffing, quality control, and all other areas of concern.
2. **Backup Programmer:** The backup programmer is at the same skill level as the chief programmer. Recognition that two heads can be better than one. They could bounce ideas off each other. If a problem was encountered, one would research the solution while the other continued with other areas of the design. Provided protection to company in case the chief programmer became incapacitated or left.
3. **Program Secretary:** Today called Version Control officer. Responsible for version control, tool selection, tool creation, job control language (JCL, make files, etc.), system documentation, user documentation, and resource for other team members.
4. Project Officer: If here were sufficient personnel on the team, a project officer was assigned to relieve the chief programmer of necessary but mundane work with respect to budgets, personnel problems, and other responsibilities as assigned by the chief programmer. This is an inversion of authority lines in the conventional team.
5. Three to five additional programmers, as required to complete the project under the direction of the chief programmer.
6. Specialist such as tax consultants, lawyers, and personnel with expertise in the area for which the software is being developed. These individuals were frequently transient in nature.

**The Terry Baker / Harlen Mills (IBM Fellow) to implement the New York Times Morgue File emphasizing:**

1. **Top Down Structure Design** (detailed design prior to coding)
2. **Structured Programming** (test Edgar Dijkstra’s theories including all modules should have a single entry and single exit, read top to bottom like a book, avoid undisciplined branching -goto statements (Go-TO Considered Harmful), etcetera)
3. **Walk Through (specification, design, coding and creation of test cases: IBM Fagan, Weinberg, and others)**
4. **Incremental Testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Interaction Difficulty  (Baker/Mills) | **High Level Language Lines (PL/1)** | **Assembly** | **Total Lines** | **Today’s terminology** |
| **Hard** | 5,034 |  | 5,034 | **High coupling, low cohesion.** |
| **Standard** | 44,247 | 4,513 | 58,760 | **Medium coupling and cohesion** |
| **Easy** | 27,897 | 1,633 | 29,530 | **Low coupling and high cohesion.** |
| **Total Delivered** | 77,178 | 6,146 | 83,329 |  |

**Summary:**

1. 132 man-months of effort completed in 22 months (11 man years)
2. Traditional team implemented projects were producing 3 to 6 lines of code per man day in the early 1970’s.
3. The file processing system was delivered 1 week after unit coding. It passed a one week acceptance test without a single error. It ran 20 months (nearly 2 years) before the first error was detected (today - agile teams, use part of system while developing rest).
4. In the first 13 months of online operation, only one program error resulted in a system failure.
5. The maintenance history was only average.
6. Apparent maximum project size limitation of about 100,000 lines of code. Use multiple teams for larger projects.
7. Computer time was nearly a constant from about the 9th through the 22th month.

Reported it IBM Systems Journal and in Datamation, December 1973, pages 58-61.

**Productivity**

|  |  |
| --- | --- |
| Organization | Source Lines / Programmer Day |
| Unit design, programming, debugging, and test | 65 |
| All professional staff | 47 |
| With library support (Version Control) | 43 |
| Entire Team including secretarial help | 35 |

Traditional team implemented projects were producing 3 to 6 lines of code per man day in the early 1970’s (programmers only).

\*\* Mills had a reputation as a “super programmer - code faster than a speeding bullet, leap reams and reams of output in a single bound).” Could mere mortals produce these kinds of results?

## Project Efficiency Goals: Sell **“one” versus 17 million copies of MS-Word**. Create "perfect" versus “just good enough” philosophy.

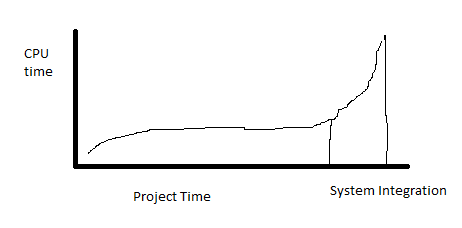
**Sky Lab (response to Russian space station) Mission System for Training Astronauts**

1. 400,000 lines of code
2. completed in a two year period
3. delivered on schedule despite 1,200 formal changes to the requirements, and cuts in man power and budget towards the end of the project. (**1,200 changes / 24 months = 50 formal changes/month** or **50 changes per month / (5 days \* 4 weeks per month) = 2.5 formal changes per day**)!

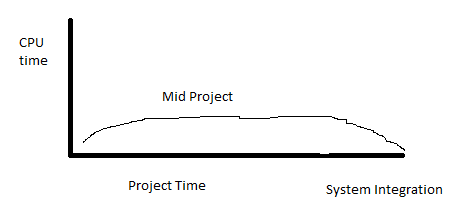
Computer time was nearly a constant from the 9th through the 24th month. The same phenomenon was noted in the New York Times Morgue File Project. In typical projects, computer time typically increased slowly or leveled off till about two thirds of the project was completed. As unit testing was completed in the final stages, computer time would drop. After all unit testing, the modules were typically combined on traditional projects. The result was a tremendous spike as this is when misunderstandings between designers, programmers, and users were uncovered. The result was a high need for man power, computer time, and budget in traditional projects once the system testing (shake and bake) commenced.

Neither the New York Times Morgue File or Sky Lab projects experienced overtime peaks at the end of the project.

Traditional project resource utilization reflecting higher resource utilization during system integration as misunderstandings etcetera are discovered.



The Chief Programmer Team approach showed no major resource utilization increase during integration. Integration effectively was done throughout the project using "Incremental Testing" in conjunction with test harness and stubs. The test harness and stubs were either thrown away after serving their purpose or served as a starting point to develop the module they simulated.



**Critics Review**

1. The system is too dependent on performance the chief and backup programmers.
2. The chief programmer team encourages a tendency towards “role” specialization.
3. The chief/backup programmer makes all assignments and gets best jobs. Typically the chief programmer and backup programmer code the upper level modules responsible for coordination and control. Junior/other programmers tend to get the lower level modules.
4. The backup programmer, librarian, and other programmers may feel their work is less appreciated.
5. The chief programmer gets all or most of the credit.

**Software Development/Programming is too important to be left to junior personnel. Traditional teams must refute the charge they encourage development by less qualified personnel.**

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

**The Terry Baker / Harlen Mills project to implement the New York Times Morgue File emphasizing:**

1. **Top Down Structure Design** (prior to coding)
2. **Structured Programming** (test Edgar Dijkstra’s theories including all modules should have a single entry and single exit, goto, etc.)
3. **Walk Through**
4. **Incremental Testing**

In 1972 Edgar Dijkstra was awarded the ACM Turing Award, the Nobel Prize of computer science. "The success of projects such as the New York Times Morgue File (thought to hinge on use of structured programming – "goto" considered harmful) weighed heavily in the balloting. A primary goal of Brooks” was to test Dijkstra’s structured principles. Dijkstra’s acceptance speech was titled:

**“The Humble Programmer”**

It was anticipated that Dijkstra would review the reasons “structured programming” helped to make these large projects so successful. He shocked the audience by stating: The success of the New York Time project cannot be attributed in whole to “structured techniques.”

**“The problem is as much one of organization as of technology and the chief programmer team is primarily an organizational solution.”**

The success was depended on:

1. A complete design prior to coding (reduced duplication, etc.).
2. Structured walk through (found errors early rather than late).
3. Structured programming.
4. Incremental testing (greatly reduced debug time).
5. The team structure re-introducing the most skilled employees back into the development process!

In the Mythical Man-Month, Brooks stated, “A projects milestones must be easily identifiable, easily verifiable, and razor sharp". The chief programmer team approach supplies these milestones. If there are 100 modules of approximately the same size and difficulty, 80 have been completed and integrated into the testing framework, then the project is approximately 80% complete. In a traditional project not using incremental testing (systems integration after completing/testing modules individually), 80% complete modules does not indicate 80% complete.

Fuzzy milestones only result in slippage and self deception!

**Software Quality Assurance (SQA)**

**and Software Reviews**

***Industry studies at TRW, Nippon Electric, Mitre Corp and others indicate that design activities introduce between 50% and 65% of all errors (defects) during the development phase of software projects. Formal review techniques have been shown to be up to 75% effective in uncovering design flaws prior to coding.***

***Reviews should be conducted to:***

**Point out improvements in the product of a single person or team.**

**Confirm those parts of a product in which improvement is either not needed or desired.**

**Help to achieve a more uniform, predictable level of quality than can be achieved without reviews.**

**Reviews are desirable after:**

**System specification to get client approval.**

**Design.**

**Coding.**

**Development of test cases.**

**T. C. Jones, "Programming Productivity," McGraw-Hill, 1986.**

**"Implementing Software Inspections," course notes, IBM Systems Sciences Institute, IBM Corporation, 1981.**

**M. Fagan, "Design and Code Inspections, and Process Control in the Development of Programs," IBM Technical Report 21.572, December 1974.**

**Daniel P. Freedman and Gerald Weinberg, "Walkthroughs, Inspections, and Technical Reviews Evaluating Programs, Projects, and Products," 3rd edition, 1982, Little Brown and Company, ISBN 0-316-292826.**

***Formal Technical Reviews or***

***Structured Walk Throughs***

***Walk-throughs (Gerald Weinberg) are most productive when:***

**The material to be discussed is distributed to the attendees for review two or three days before the walk through. *Management must support the position that all attendees do their home work prior to the walk-through to avoid wasting the group’s time.***

***The number of attendees is limited to four or five and the session to about one hour.* The project manager may be barred from the walk-through if they are not technically competent or it is felt their presence may inhibit the frank and open exchange of ideas. Education of junior members is a goal of the review process.**

**In some cases, it may be necessary to include an impartial mediator to resolve disputes. Raise issues, don't resolve them. Watch your language, be humble, avoid critical negative phrasing. Stick to the issues. There is always some danger an overly aggressive group will be detrimental to a designer or programmer who is not very sure of their work. On the other hand, an overly aggressive individual may overwhelm the suggestions or criticisms of the group.**

**Technical reviews should specifically "avoid" redesigning systems or programs. The purpose of the meeting is to verify the specification/design/code/test cases accomplish the desired purpose and identify problems. Spontaneous group designs almost always end in failure.**

**Formal notes should be taken (signed/approved by the group) and the individual or group that is the subject of the inspection should be expected to respond to the groups suggestions in a reasonable time.**

***The entire "group" is responsible for any flaws or bugs in the final product.***

**M. Fagan’s Experiments at IBM (1974, 1976)**

Technical Inspections are desirable at:

Level 0: Statement of Objectives

Level 1: After the Design

Level 2: After Coding

Fagan only had sufficient funds to support inspections in two levels. He chose to do the inspections at level one and two. The conclusion:

**Technical inspections conducted at levels one and two save about one programming month per 1000 non-comment source statements. The inspections more than pay for their expense in terms of development time and product quality!**

Structured Reviews are widely used by contractors on the space shuttle, Space Station, White Sands, other government and civilian projects. They are credited with making very large contributions to the quality of the reviewed projects. Today use of technical review is associated with high quality in all engineering and technical fields. It is also applicable to the managerial decision making process and most if not all other areas in which complex issues must be examined leading to important decisions. Lack of formal review tends to produce low quality results.

**Peer Review and Ratings**

1) Peer review is based on the idea that the members of a group are far more familiar with the contributions of members due to their close association. The military and many companies have found it to be very effective in rating performance and determining suitability to lead, frequently more accurate (better predictor of performance) than:

A) Objective test (military, civil service, RN vs. LVN Vs. RN/LVN)

B) Interviews

C) IQ Tests

1. Education Levels

F) Administrator Ratings

2) Peer reviews are not adversely affected by untrained evaluators and friendships play only a minor role.

3) Peer reviews are affected by:

A) Racial Prejudice, tendency to rate candidates of the same race higher.

B) Younger, less senior, more educated (silver spoon) are subject to lower ratings by peers than managers.

4) Peer Review only works if the anonymity of the rater is assured (confidentiality). Those rated benefit from summary information when the group doing the rating is sufficiently large to mask individual ratings.

As an example, have two different development teams anonymously exchange code for rating. Each work is rated by several members of the rating group. Results are presented to the individual in summary fashion.

|  |  |
| --- | --- |
| Question | No Neutral Yes |
| 1) reasonable name | 1 2 3 4 5 6 7 8 9 10 11 |
| 2) sufficient comments |  |
| 3) pleasing format |  |
| 4) low level logic comprehensible |  |
| 5) high level design apparent |  |
| 6) methodology applied correctly |  |
| 7) algorithm a good choice |  |
| 8) overall program easy to comprehend |  |
| 9) easy to modify |  |
| 10) machine/OS independent |  |
| 11) compiler independent |  |
| 12) would you be proud to have written |  |
| 13) sensible data structures |  |
| 14) would you find it hard to improve |  |
| 15) high cohesion |  |
| 16) low data coupling |  |
| 17) low control coupling |  |
| 18) proper use of language features |  |
| 19) highly factored |  |
| 20) proper use of private/local, global/public |  |
| 21) appropriate use of fan-in/fan-out |  |
| 22) appropriate re-use of existing code |  |
| etcetera |  |

Summary for individual being rated:

|  |  |  |  |
| --- | --- | --- | --- |
| Question | Rater  1 2 3 4 5 | Average | Average all being rated |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| etcetera |  |  |  |

Field trials suggest peer rating are productive, enjoyable, non-threatening experiences.**Joint Application Development (JAD):**

**JAD was originally developed by IBM Canada in the 1970's as a means for *bringing users and systems analysis's together for intensive, highly productive mediated sessions to elicit the requirements* for a new system. CASE tools have revived the concept.**

**Rapid Application Development (RAD)**

**A modern variation of JAD. Usually described as a *combination of JAD sessions to determine user requirements quickly, as well as CASE tools, prototyping techniques, Rambo-style SWAT teams, and a formal software development methodology* to implement those requirements quickly.**

**As a combination of tools and techniques, RAD has a lot of potential. The individual techniques used alone should not be expected to perform adequately on a regular basis.**

**Some people just see this as prototyping. It is *particularly useful when the user is unsure of his or her real requirements, the user interface, algorithms required, or the desired/required inputs and outputs*.**

**It is expected that the use of CASE tools in conjunction with JAD will result in the evolution of *"groupware"* tools to facilitate cooperative work activities in teams.**

**Tiger Teams (put out fires)**

**Agile (scrum/meetings): Typically develop the product in usable parts while the rest of the project is under development.**

**Patterns (versus generics/templates)**

Agile Methodologies

**Many variations of agile programming are popular including Extreme Programming (XP) and Scrum. Many agile projects use the Rational Unified Process (RUP – IBM) as a development framework. XP was introduced by Kent Brock at the Chrysler Corporation for the C3 project emphasizing small teams working in the same room to encourage communications. XP does not fit well with classical approaches to software development. It emphasizes building products in small iterative steps with minimal documentation. The core of XP may be summarized as follows:**

1. **Frequent communications between the team members and customer (likely on site) are required/encouraged.**
2. **Simplicity in both code and design is emphasized in each iteration.**
3. **The small iterations provide constant customer feedback. Unit tests and continuous product integration provide feedback to the individual developer (or developer pair).**
4. **Emphasize the requirement to make hard but necessary decisions.**

**Five fundamental values are at the core of XP ( Extreme Programming):**

1. **Rapid feedback provided by short development iterations/releases using pair programming, unit testing and continuous integration.**
2. **Try the simplest approach attempting to minimize coupling while maximizing metrics like cohesion between iterations. Do not be overly concerned about considerations that that may or may not occur in the future.**
3. **Emphasize small incremental changes the sum to major changes. This “refactoring” is applied to design, coding, planning and team composition. The XP process is expected to evolve in this manner.**
4. **Embrace change by preserving options for the future while solving your most pressing problems.**
5. **Encourage all team members to do their highest quality work at all times.**

**XP emphasizes 12 key practices:**

1. **Planning quickly to determine the key features for the next release.**
2. **Try to get a partially working system as quickly as possible emphasizing short releases to form a platform for evaluating the future.**
3. **Instead of a formal architecture, use a simple metaphor for the system so everyone understands the goal.**
4. **Keep the design and system as simple as possible emphasizing appropriate desirable development metrics form iteration to iteration.**
5. **Ensure testing is done continuously automating as much of the testing process as possible. All code must be subject to unit testing. Write tests prior to the code. Ask the customer to to write functional development test to verify when features are complete. Continue to use the test throughout the development process – test-driven development.**
6. **Improve the system via design improvement/refactoring. Restructure the system for improvement to remove duplication, improve communications, simplify or add flexibility. The design is not completed prior to coding but rather evolves and improves over time.**
7. **Practice pair programming. All production code is written by two programmers working at the same work station or facility. The improvements in quality typically more than make up for the dual efforts.**
8. **Emphasize collective ownership of the product.**
9. **Emphasize continuous integration of the system. It should be rebuilt and tested every time a new work product is completed.**
10. **Work at a sustainable pace, e.g., 40 hours per week. Do not emphasize use of overtime.**
11. **The customer should be on-site and participate in the decision making process.**
12. **A set of standards for coding, documentation etcetera should be adopted and followed by the entire team.**

**These goals are not always easy to accomplish. In general it is difficult to have an on-site customer and retain high cohesion with minimal coupling between iterations. Customers frequently have other responsibilities. Limiting work to 40 hours per week is difficult without a very enlightened management. The rewards achieved by many industry teams however make the process very attractive.**

**This is a dramatically reduced introduction to XP. Each of the numbered bullets above would expand to at least a few pages.**

**As a partial example, assume you have been commissioned to create a newspaper morgue file. Essentially such systems store both printed stories and the research behind them. The information is typically indexed in multiple ways, by author, range of dates, key words and subject area. When a reporter is writing a new story, they can research the morgue file for background information and possibly leads to expand on their current story. At a minimum the morgue file will require a commercial or tailored database system, query language or screens to help a user find the desire information using all retrieval methods. You would want to automate the addition of new information to the system as much as possible.**

**Once the ER diagrams have been created for the database, we might have one group create the programs to allow the customer to do the initial data entry. Incorporating new data can wait till latter. Indexing will also be minimal. Data clustering algorithms to find potentially related materials will also not exist in the first round. Another team can be working on the query language using the available indexing. This will allow the customer to start the data entry and generate some reports/queries. Back up procedures will probably consist of making a copy of the entire data system initially. Latter more automated and sophisticated means of protecting the system from loss will have to be constructed. If only the customer can access the system locally we can put off some of the security till a latter iteration. Future iterations will add access methods and clustering to the data, associated retrieval methods, more sophisticated and easy to use reporting methods, etcetera.**

**Characteristics of Agile Versus Traditional Processes**

|  |  |  |
| --- | --- | --- |
|  | **Agile/Extreme Programming (XP)** | **Traditional** |
| **Requirements** | **Assumes requirements will change at each iteration. Constant user involvement during project as opposed to once at start.** | **Assumes change will not occur or be minimal. Changes after the design or during implementation will be costly.** |
| **Design** | **Informal and iterative.** | **Formal, done prior to implementation with known requirements.** |
| **User Involvement** | **Frequent and crucial throughout entire project.** | **During requirements definition and acceptance testing.** |
| **Documentation** | **Minimal, informal. Source code is ultimate documentation.** | **Formal documentation during all phases.** |
| **Communication** | **Informal and continuous.** | **Relies mainly on documents, memos, formal reviews, and meetings.** |
| **Process Complexity** | **Relatively low. Restricted to small teams (10 of less). Does not scale well to larger projects.** | **High. Rational Unified Process (RUP), CMM etc. may have over 100 artifacts, 30 roles, and multiple phases.** |
| **Total Overhead** | **Low. Normally does not scale well to larger projects. Heavy reliance on teamwork, customer interaction, cultural clash with traditional methods. Iteration results in improved ability to handle change, early use by customer, high acceptance rate. Customer sees partially usable systems much earlier.** | **Relatively high but can be scaled down for smaller projects.** |
| **Scope/Size** | **Small projects typically limited to teams of 10 or fewer people.** | **Best suited to larger projects; scales well to larger projects (RUP, SEI/CMM).** |
| **Critical Systems** | **Not suitable for life critical systems without adaptation.** | **Used for mission critical systems with no or minimal modifications.** |
| **People** | **Very suitable for use with “team” players who are good at both design and coding. XP requires strict adherence to specific practices for success.** | **Can be used with most reasonable individuals,; does not require straight team playing; does define many specific roles.** |
| **Culture** | **Best suited to small co-located companies with relaxed cultures. Does not work well in projects involving multiple companies as customers cannot provide the required level of continuous feedback.** | **Works with large groups separated geographically and more formal cultures.** |
| **Stability** | **Copes well with changes in requirements and environments.** | **Best when environment relatively stable with few changes in requirements. Adoptable to other environments.** |