**M S Ramaiah Institute of Technology**

(An Autonomous Institute, Affiliated to VTU)

MSR nagar, MSRIT post, Bangalore-54

A ***Project Plan*** on

**Android based Monitoring Human Knee Joint Movement Using Wearable Computing**

Under the guidance of

Mr. M. Mallegowda

Submitted by

Sanjana K. S 1MS12CS098

Vidit Jain 1MS12CS127

Vignesh P 1MS12CS128

Vishal H 1MS12CS132

*In partial fulfillment for the award of the degree of*

# *Bachelor of Engineering in Computer Science & Engineering*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**M.S. RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

[www.msrit.edu](http://www.msrit.edu), **May 2015**

**PROCESS MODEL: EXTREME PROGRAMMING**

**NEED FOR XP**

While developing software, the purpose is not only to develop software to satisfy the needs of some users or clients, but we want that the project be done in low cost and cycle time, and deliver high-quality software. In addition, there could be other constraints in a project that the project may need to satisfy. Hence, given the constraints of the project, we would like to employ the process model that is likely to maximize the chances of delivering the software, and achieve the highest Q&P. Hence, selecting a suitable development process model for a project is a key decision that a project manager has to take.

In our project we would be using extreme programming software development process model as the requirements might change and further modifications are to be done to the software in iterations. Further the coding would be done by Vignesh and Vishal simultaneously by distributing the work and also test driven development will be used where test cases are written first and later the actual code is developed. This ensures that the process of coding becomes quicker as only slight modifications are to be made, if any, to the code. Thus it’s quite befitting that we need to use extreme programming software development model.

**INTRODUCTION**

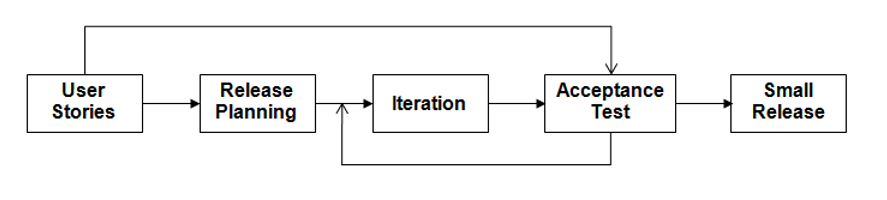
Agile development approaches evolved in the 1990s as a reaction to documentation and bureaucracy-based processes, particularly the waterfall approach. Agile approaches are based on some common principles, some of which are

* Working software is the key measure of progress in a project.
* For progress in a project, therefore, software should be developed and delivered rapidly in small increments.
* Even late changes in the requirements should be entertained (small-increment model of development helps in accommodating them).
* Face-to-face communication is preferred over documentation.
* Continuous feedback and involvement of customer is necessary for developing good-quality software.
* Simple design which evolves and improves with time is a better approach than doing an elaborate design up front for handling all possible scenarios.
* The delivery dates are decided by empowered teams of talented individuals (and are not dictated).

Many detailed agile methodologies have been proposed, some of which are widely used now. Extreme programming (XP) is one of the most popular and well-known approaches in the family of agile methods. Like all agile approaches, it believes that changes are inevitable and rather than treating changes as undesirable, development should embrace change. And to accommodate change, the development process has to be lightweight and quick to respond. For this, it develops software iteratively, and avoids reliance on detailed and multiple documents which are hard to maintain. Instead it relies on face-to-face communication, simplicity, and feedback to ensure that the desired changes are quickly and correctly reﬂected in the programs.

**EXTREME PROGRAMMING**

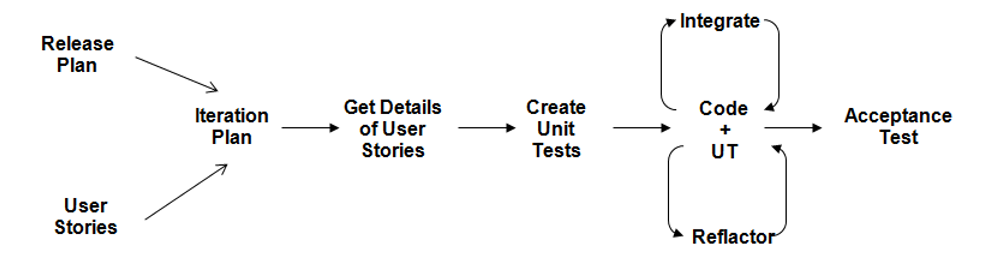
An extreme programming project starts with user stories which are short (a few sentences) descriptions of what scenarios the customers and users would like the system to support. They are diﬀerent from traditional requirements speciﬁcation primarily in details—user stories do not contain detailed requirements which are to be uncovered only when the story is to be implemented, therefore allowing the details to be decided as late as possible. Each story is written on a separate card, so they can be ﬂexibly grouped. The empowered development team estimates how long it will take to implement a user story. The estimates are rough, generally stated in weeks. Using these estimates and the stories, release planning is done which deﬁnes which stories are to be built in which system release, and the dates of these releases. Frequent and small releases are encouraged, and for a release, iterations are employed. Acceptance tests are also built from the stories, which are used to test the software before the release. Bugs found during the acceptance testing for an iteration can form work items for the next iteration. This is shown in Figure1.



**Figure1. Overall Process in XP**

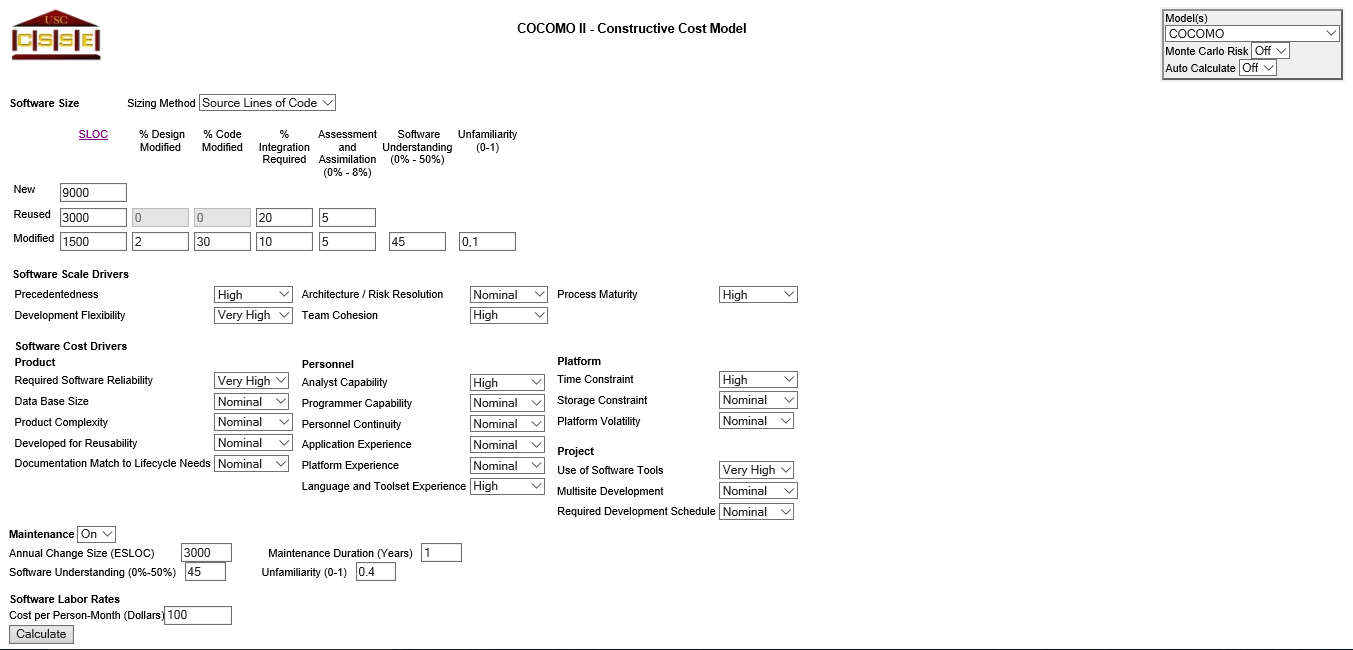
Development is done in iterations, each iteration lasting no more than a few weeks. An iteration starts with iteration planning in which the stories to be implemented in this iteration are selected—high-value and high-risk stories are considered as higher priority and implemented in early iterations. This is shown in Figure2. Failed acceptance tests in previous iteration also have to be handled. Details of the stories are obtained in the iteration for doing the development. The development approach used in an iteration has some unique practices.

* First, it envisages that development is done by pairs of programmers (called pair programming).
* Second, it suggests that for building a code unit, automated unit tests are written ﬁrst before the actual code is written, and then the code should be written to pass the tests. This approach is referred to as test-driven development.
* Third, as it encourages simple solutions as well as change, it is expected that the design of the solution devised earlier may at some point become unsuitable for further development. To handle this situation, it suggests that refactoring be done to improve the design, and then use the refactored code for further development. During refactoring, no new functionality is added, only the design of the existing programs is improved.
* Fourth, it encourages frequent integration of diﬀerent units. To avoid too many changes in the base code happening together, only one pair at a time can release their changes and integrate into the common code base.

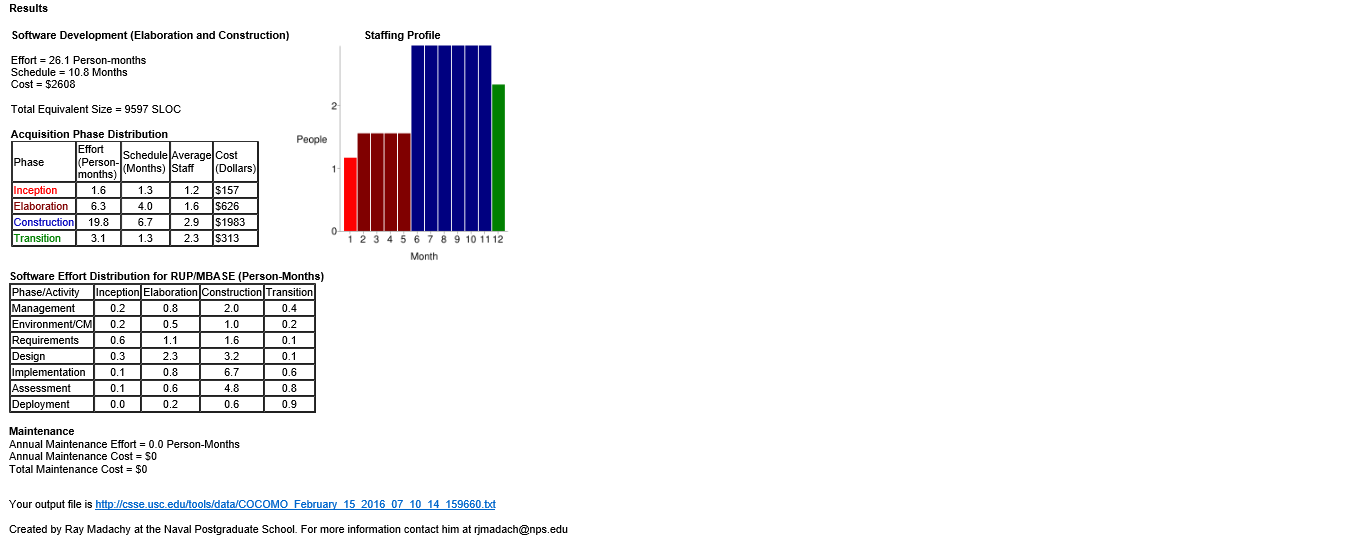


**Figure 2.An Iteration in XP**

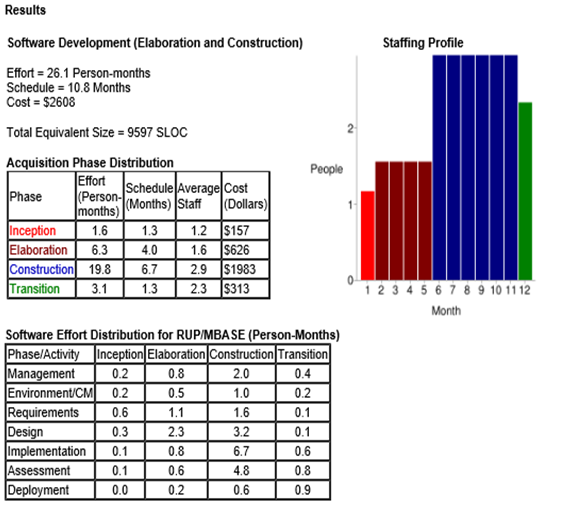
**EFFORT ESTIMATION USING COCOMO MODEL**



**Figure 3. Online COCOMO Model Page1**



**Figure 4. Online COCOMO Model Page2**



**Figure 5. Results**

**RISK IDENTIFICATION AND MITIGATION**

**INTRODUCTION**

Risk is deﬁned as an exposure to the chance of injury or loss. That is, risk implies that there is a possibility that something negative may happen. In the context of software projects, negative implies that there is an adverse eﬀect on cost, quality, or schedule. Risk management is the area that tries to ensure that the impact of risks on cost, quality, and schedule is minimal.

Risk Mitigation, within the context of a project, can be defined as a measure or set of measures taken by a project manager to reduce or eliminate the risks associated with a project. Risks can be of various types such as technical risks, monetary risks and scheduling-based risks. The project manager takes complete authority of reducing the probability of occurrence of risks while executing a project.

**TOP 5 RISK ITEMS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **RISK ITEM** | **RISK MITIGATION TECHNIQUES** |
| 1. | Developing the wrong software functions. | Early user manuals, User surveys and developing the software in iterations (prototyping). |
| 2. | Developing the wrong user interface. | Task Analysis, User Characterization, Scenarios and Prototyping. |
| 3. | Updating objectives. | Releasing the software in small increments. |
| 4. | Lack of technical and management skills. | Adequate training and also allocating tasks to people based on their capabilities and strengths, enhanced communication between team members |
| 5. | Use of non effective components. | Clearly understanding the requirements and analyzing the scenarios, requirements scrubbing, cost benefit analysis. |

|  |  |
| --- | --- |
|  |  |