Assignment

NAME : M.JASMAN

SUBJECT : RAPID APPLICATION DEVELOPMENT

DEPARTMENT: COMPUTING & INFORMATION SYSTEMS

DUE DATE : 2020.O4.10

**Best practices in RAD**

1. **TIME BOXING**

* Rapid Application Development(RAD) technique that:
* Focuses on schedule
* Imposes a fixed time limit of a project
* Focuses on critical requirements
* Further scope creep
* Does NOT work for all types of projects
* Very large projects
* Projects in core business areas
* Is often deployed with other RAD techniques, such as Evolutionary prototyping, CASE, SWAT and JAD.
* Basic for many RAD methodologies
* SCRUM
* RUP

1. **JOINT APPLICATION DEVELOPMENT(JAD)**

* JAD is a requirements-definition & user interface design methodology and attend offsite meetings to work out a system’s details.
* JAD focuses on the business problems rather than technology problems.
* Applicable to the development of business systems.
* It produces its savings by shortening the elapsed time required to gather a system’s requirements and by gathering requirements better, thus reducing the number of costly, downstream requirement changes.
* Its success depends on the effective leadership of the JAD session; on participation by key end-users ,executives ,and developers
* Achieving group synergy during JAD Sessions.

1. **PRINCIPLED NEGOTIATION**

* Clarifying expectations and identifying exactly what is needed to set the project up for success.
* Removes problems from people and seeks solutions outside of those problems
* It can be used any kind of projects**.**

1. **REQUIREMENTS SCRUBBING**

* This is using the pareto80/20 role to
* After you create a req.specification, you go over it with a fine tooth comb:
* Eliminate all requirements that are not absolutely necessary.
* Simply all requirements that are more complicated than necessary.
* Substitute cheaper options for all requirements that have cheaper option.

1. **REUSE**

* 90% reduction in development time
* Greatly increased quality
* Requires the right kind of culture
* What could the culture be?

1. **OUTSOURCING**

* There are folks outside who can do it better faster and less money than the folks inside.
* Outside sources may have solved the problems many times before and therefore be much further down on the learning curve
* Outside sources may be able to extensively reuse.
* Save development cost.

1. **MEASURAMENT**

* Doesn’t produce results within the span of one project, but over several projects, processes and practices are improved.
* There is tendency to measure everything just in case you need it.
* A better practice is to allow measurements to be driven by goals, questions, and metrics.
* Measurement should provide progress visibility
* M should focus people’s activities
* Improve morale
* M Help set realistic expectations
* M Lays the groundwork for process improvement
* Time-after-time Gold raft finds companies that are failing because they are measuring and rewarding the wrong things
* Is the antidote to the common problems of poor estimates, poor scheduling, and poor progress visibility?
* Has the potential to reduce the duration of the project schedule, improve progress visibility, and reduce schedule risk.

1. **MANIATURE MILESTONES**

* Improves visibility
* Provides fine-grain control
* Improves motivation
* Reduces schedule risk
* Never let a developer go DARK!!!
* Are obviously milestones that are between major milestones
* Provides visibility and confidence that major milestones will be reached.
* EVERYONE BECOMES AWARE THAT A PROJECT IS GOING TO SLIP MUCH SOONER.

Advantages:

* Can be used throughout the development lifecycle, not just the construction phase
* Works well with just about any kind of software development
* Provides developers with a steady sense of accomplishment
* Is a fine-grained approach to project tracking and control that provides exceptional visibility into a project’s status?
* Eliminates the risk of uncontrolled undetected scheduled slippage
* Works well when used with the daily build and smoke test

1. **GOAL SETTING**

* Potential for reduction from nominal schedule-very good
* Chance for first – time success –good
* Chance of long –term success –very good
* Considered GOOD OVERALL in terms of creating a shorter schedule.
* Makes use of the fact that human motivation is the single, strongest contributor to productivity.
* In goal setting, a project manager or customer simply tells developers what is expected of them.

1. **RAPPID DEVELOPMENT LANGUAGE & TOOLS**

* Can improve productivity greatly
* Produce their savings by reducing the amount of construction needed to build a product.
* It helps to :
* shorten the construction cycle
* lack first-rate performance
* constrain flexibility
* limited –distribution custom software than to system software

1. **DAILY BUID AND SMOKE TEST**

* Is treated as the heartbeat of the project
* Uses an automated build tool such as make in VB
* On large projects someone on the term has responsibility for conducting the daily and smoke test
* DMST’s are performed in the evening and if successful released the next morning for use by the team
* You build the product every day and test it minimally every day.
* If the build to create an executable doesn’t work. It is considered broken and becomes the highest priority of the term to get fixed.
* A clean build is one in which all source files compile to object modules
* All files link successfully
* Smoke test is passed

1. **INSPECTIONS**

* Detection of errors early
* Avoids costly downstream work
* Can be used on both development and maintenance
* There are formal technical reviews in which participants in the review are well-trained in review practices and assigned specific roles to play.
* The roles played during the review meeting help to stimulate discovery of additional errors.
* Have been found to be much more effective in finding errors than executing testing.
* Both in percentage of total defects found and in time spent per defect.

1. **EVOLUTIONARY DELIVERY**

* Delivering selected portions of the software earlier than would otherwise be possible , but it does not necessarily deliver the final software product any faster
* Ability to change product direction mid-course in response to customer requests.
* Improved product quality
* Reduce cost size

1. **EVOLUTIONARY PROTOTYPING**

* Developing selected portion of the software earlier than would otherwise be possible.
* Provide ability to change product direction mid –course in response to customer requests.
* Used successfully on in-house business software and shrink-wrap software.
* It can lead to improved product quality.
* Reduce cost size
* Distribution of development and testing resources.
* Establish a stable, static core architecture for the product, the application.
* Deliver the customer’s first understanding of the problem early.
* Support to early delivery
* This gets some functionality into the hands of the customers or end user at an early date.

1. **THROWAWAY PROTOTYPING**

* Develop the prototype quickly
* Test it
* Throw it away
* Take what you learned and use it to develop the final version of the software.
* Throwaway prototyping practice can accelerate projects based on traditional life cycle models
* Prototype implementation uses programming languages or development practices or both that are much faster than the target languages or practices.

1. **USER-INTERFACE PROTOTYPING**

* Development time is drastically reduced.
* Needs user requirement throughout the life cycle of the product.
* Most productivity with fewer people.
* Only suitable for projects which have a small development time.
* Time between prototypes and interactions is short.

1. **DESIGNING FOR CHANGE**

* Such changes make maintenance much easier
* Means good program structure and high quality code
* Supports evolutionary/incremental/versioned delivery

-giving your customers a piece of functionality at a time

* Enables changes late in the project to be effected easily, rapidly.
* Change has happen because of market conditions, the customer’s understanding of the problem changes, or the technology changes.
* Because it is very difficult to get requirements right the first time
* Customers don’t know what they want
* Requirements modeling has improved requirements determination, but still there are many problems.
* Modest potential reduction in nominal schedule
* No improvement in progress visibility
* Decreased scheduled risk
* Good chance of first/time success/excellent chance of long –term success.

1. **CHANGE CONTROL BOARD**

* It helps rapid-development by ◦ raising the visibility of feature creep ◦ reducing the number of uncontrolled changes to the product