PRESENTATION

T15

Thursday, March 9, 2000
1:30PM

RELENTLESS APPLICATION DEVELOPMENT

Linda McInnis and Marco Ocaña

Millennium Pharmaceuticals, Inc.

International Conference On
Software Management and Applications of Software Measurement
March 6 – 10, 2000
San Jose, CA

Relentless Application Development

Linda McInnis & Marco Ocaña

Millennium Pharmaceuticals, Inc.

- Relentless Application Development is our term for continuous, rapid development and deployment of mission critical applications in highly changeable environments.

Why relentless?

- Biotech industry (the sequencing of the human genome is being finished and it's a race to patent novel sequences)
- High throughput laboratory technology and analysis methods are constantly being introduced.
- Fierce competition to deliver large quantities of data for drug discovery
- To fulfill our corporate mission,

"To Transcend the Limits of Medicine"

Our Attitude

- Suspicious of big process
- Wary of “experts” and silver bullets
- Make it easy for people to do the right thing
- Success of the project is everyone’s responsibility
- Practice makes perfect – not every project follows the same process

What are appropriate projects for relentless development?

- Extension of existing applications – not new applications
- Internal applications–not shrink wrapped
- Continuous feature enhancement models

Before you begin…

- Configuration Management and build procedures, Installers
- Development and Test environments
- Project Management (Project Tracking)
- Communications Technologies (web, email, groupware)
- Defect Tracking

Initial Planning

- Identify stakeholders
  - Sponsor
  - Customers
  - Others
- Specify success criteria for the project
- High Level requirements
- Identify the people resources that will be needed. Identify roles and success criteria
- Revisit the list of deferred bugs.

Scheduling

- Can’t get it completely right from the start
- Tasks should be broken down to smaller tasks no bigger than 5 days
- Mini-milestones all along the way
- Each task has a go/no-go completion criteria
- Allow some slack time to allow for contingencies
- Draft schedule should include holidays, vacations any known company events.

Risk Management

- Whatever can go wrong…
- Optimism is overrated
- Most problems can be foreseen and planned for
- Create and maintain a list of risks
- Involve people in the mitigation plan
- It’s an on-going battle

Most Common Schedule Risks*

- Feature creep
- Requirements or developer gold plating
- Shortchanged quality
- Overly optimistic schedules
- Inadequate design
- Silver bullet syndrome
- Research oriented development
- Weak personnel
- Contractor failure
- Friction between developers and customers

*Steve McConnell, *Rapid Development*

Execution – actually running the project

- Keep an issues list
- Manage to plan
- Frequent check ins (at least a couple of stand up 15-minute meetings)
- Bug Triage – early and often

People and Teams

- Right set of skills, set of reasonable people
- Need to know what people are capable of doing.
- Team problems-take care of them right away
- Make sure everyone knows what they are doing when and what the success criteria is
- No amount of planning can make up for a team that does not gel

Quality Assurance

- QA personnel involved from the beginning.
- To automate or not to automate?
- Bug tracking system.
- Developer unit tests.
- QA does GUI and integration and system tests.
- Triage early and often.

Documentation

- Can make or break your product with users
- Involve them early
- Involve them in prototyping, spec writing, requirements
- Don’t print documentation – make it web-based and you avoid long delays in getting things printed

The End Game

- Post-mortem
- Final bug resolution
- Celebration

10 Lessons Learned

- Pay Attention – projects don’t run themselves – Active Project management
- Review the work of new people, particularly if they are inexperienced. Avoid surprises
- Do not skimp on analysis unless you are willing to rework things again and again.
- Give people space to explore and play, but make sure that first things come first: the project.

10 Lessons Learned, continued

- If something is not working, let it go. And the sooner you let it go, the better.
- It is important to engage everybody's mind on the project. Don't try to do all the thinking.
- Having documentation enables you to bring in more people to help out.
- Know the capabilities of the people you are working with (strengths and weaknesses)

10 Lessons Learned, continued

- Every task in the project plan should be associated with a deliverable and every deliverable should have a task that it is assigned to, no matter how trivial.
- The earlier you can start testing, the better.

Relentless Application Development

Linda McInnis & Marco Ocaña
Millennium Pharmaceuticals, Inc.
640 Memorial Drive
Cambridge, MA 02139

Relentless Application Development is our term for continuous, rapid development and deployment of mission critical applications in highly changeable environments.

- Time sensitive development
- Rapidly evolving new technology and methods
- Computing technology is changing very rapidly giving more power for less money

What is the environment?

The environment we work in is biotechnology where every thing is time driven and time-boxed because of the potential for large gain. For example, our company profits largely by creating research programs that result in the discovery of drug or gene therapy targets (patenting genetic structures). We are pushed on by a race to the finish for the discovery of novel genes and impending deadlines (the sequencing of the human genome project is finishing ahead of schedule and those that can find, understand and patent a sequence may be financially ahead).

What we do in the Millennium Informatics group is to build software research applications that aid scientists in doing analysis. As you can imagine, to the swiftest go the spoils. Our mission is to get scientists the tools they need before anyone else and the data must be reliable and accurate. We are rather unique in that we use traditional rapid application development tools and techniques but are continuously re-inventing or extending our processes. As a result, we have found certain elements to be key in being relentless in getting new feature, tools, and models out to our users. These are listed in this paper.

What are appropriate Relentless Application Development projects?

The best candidates for using this process are

- Extending existing applications not brand new applications
- Internal applications – not shrink wrapped
- Continuous feature enhancement

We recommend this approach because new applications really require more time to design than extending an existing model particularly if the application is a foundation to future large-scale development. This approach also works well for internal applications versus shrink-wrapped because there is additional planning overhead for getting materials to manufacturing, printing manuals, etc.

Our Attitude

This is a pragmatic approach used in real projects not concocted by process “gurus”. We are trying to make this a down-to-earth, realistic way to approach projects. One size doesn’t fit all and some techniques make sense in some projects and not on others.

Example: Big projects require more rigor with documentation, sign-offs, reporting structures because keeping everyone on the same page is difficult.

To restate our objectives in this paper, we believe we can suggest ways for you to develop, test, document and deploy applications in as little as three months by using a combination of standard and innovative development processes to get speedy development and appropriate quality.

Before you begin

If your company will continually be producing software of some kind, you should get management support and staffing for the following functions that will allow you to build, test and manage software projects in an efficient way

- Configuration Management and build procedures, Installers
- Development and Test environments
- Project Management (Project Tracking)
- Communications Technologies (web, email, groupware)
- Defect Tracking

Configuration Management

We are using this term more loosely than industry standard because on small teams you will probably not have a department or even more than one person to do the configuration/release management, builds, procedures and installation scripts.

You should institute agreements with your developers that the product should be buildable on an every day basis. In other words, they won’t check they haven’t at least tested code in that and that they feel won’t break the build.

If you set up a system of build procedures that include running a smoke test as part of the build procedure, you can get a large return on testing investment and speed bug detection. It also enables you to head off creeping development time by keeping on schedule. You can segment component development at the developer level with clear responsibilities and write the test automation to email the developer with the faults in his component. It is such a good system that it allowed us to built and test every day and to release new components into production on a monthly basis.

Development and Test Environment

Having and maintaining a test environment is vitally important to speed and accuracy. They take time to set up and maintain. Our current situation is a client server set of applications that are large and closely coupled thus leading to a large number of interacting components. For example, verifying that changes to a database do not clobber another application's database change is an enormous environment management challenge.

We have found it helpful to use interns to maintain a clean and pristine environment in the test lab.

Bug Tracking/Change Management

Bug tracking is absolutely essential to determine the health of your development process. It is like the tests your doctor does such as blood pressure and cholesterol. It is not the whole story but merely an indicator of health or potential problems.

Put these things ahead of actually trying to build code because it becomes automatic behavior so you don't have to think about things but creating and testing code.

You should also keep in mind when trying to maximize speed that you will have to actively manage the project – we discuss this in the next section of this article.

Before you begin coding

As Stephen Covey tells us, you have to start with the end in mind. Before you set about coding you have to make sure you know what you are building, who will be doing what when

- Requirements
- Communications
- Team Building

Team building and Communications

You can have some of the greatest talent in the industry, but if the team can't pull together to get the job done, the best-planned project will fail. You can get an idea of the team dynamics early in the process by involving the whole team in the planning process using methods like the Yellow Sticky Method for Project Estimation. This is a technique effectively used in manufacturing for many years as a way of having each team member make estimates on the tasks before them and use peer review to determine the schedule. It gives a much more reasonable and accurate schedule and the team owns the schedule so they will be more likely to meet it and take responsibility for it.

For more looks at how team work together, get a copy of Tom DeMarco's book Peopleware.

Initial Planning

Planning is an iterative process. You don't just write the spec and begin building software.

To build the right product at the right time you have to involve the users but also be able to understand their perspective. For example, if you are building a financial management tool, you need to interview managers who will use the tool, but you also have to interview their assistants and upper management because they may be users who were not thought of originally as stakeholders.

We go into a project, knowing some prioritized high-level requirements (one paragraph descriptions), but not the requirements in detail. We then touch base with users to make sure that the priorities still apply. We recommend looking at requirements methodologies such as James and Susan Robertson's Requirements Analysis guidelines and their Volare model for requirements gathering. We also employ use cases, which also help testers get a jump on their test plans.

We create a preliminary schedule that includes some rough dates for the completion of a requirements document, the completion of design and coding, and the completion of integration testing. Be sure to

- Identify stakeholders: Sponsors, Customers
- Specify success criteria for the project
- High Level requirements
- Identify the people resources that will be needed. Identify roles and success criteria
- Revisit the list of deferred bugs.

Once we have an idea of what we're building, we try to put together a tentative schedule.

Scheduling

We are assuming that you are not constrained by an end date from above you in the organization. If you are time-boxed from the start you'll have to make some hard choices about what you can and cannot do on the project based solely on schedule. Scheduling backwards is one of the primary causes for a failure rate of about 69% in software projects.

We find that it takes several iterations to get the schedule right. Each schedule contains:

- Tasks should be broken down to smaller tasks no bigger than 5 days

- Mini-milestones all along the way
- Each task has go/no-go completion criteria
- Draft schedule includes holidays, vacations any known company events.

Allow enough time in the schedule so that if some one gets ill or goes on vacation, the project will not derail.

Most Common Schedule Risks

Most of you are risk managers and so this list should be quite familiar but if you have forgotten, Steve McConnell has created a great list of schedule risks in his book *Rapid Development*. Some are:

- Feature creep
- Shortchanged quality
- Overly optimistic schedules
- Inadequate design
- Silver bullet syndrome
- Research oriented development
- Weak personnel
- Contractor failure
- Friction between developers and customers

Risk Management

Risk Management is one of the most difficult areas to do well in a project simply because there are so many risks to cope with but as part of your job as a development, qa, or doc manager, you must cope with the unexpected. So when you are about to publish your schedule think of these points and add them to your list of things to be aware of:

- Have a plan for everything that can break
- Nail as much down as early as possible
- Make sure you've got a great team
- Have agreements in place about how things are done

Execution – actually running the project

You have gotten your requirements in shape, you've got a great team, you feel you know what you're doing – how do you keep the project on track?

There are some simple techniques that can keep you on track and on schedule

- Keep an issues list
- Manage to plan
- Frequent check ins (at least a couple of stand up 15-minute meetings)
- Bug Triage – early and often

Projects don't just happen, it's a careful dance of individual effort balanced by skilled management and a bit of luck. By keeping a list of issues present to the team, an individual knows what is the next challenge and can gracefully move on to solving it. Checking in frequently keeps people actively engaged and focused on issues. Bug triage gives you a sense of how healthy the project is.

Schedule frequent short check in meetings each week to review milestones for the week and another one near the end of the week to see if you are still on plan. These don't have to be formal, this is just to re-inforce what you are doing and is it on track.

People and Teams

If you want speed you have to have high bandwidth communications and cooperation, people have to get along. You have to set a tone and culture on the team that the priority is to ship great software. No amount of planning can make up for a team that does not gel, you have to get the right set of skills, set of reasonable people. Make sure everyone knows what they are doing when and what the success criteria is

Take care of team problems right away or they will crash your project.

Quality Assurance

Quality assurance can be your best defense against errors so involve them from the beginning. You can also get more coverage by having developers do unit tests as well.

You need to be able to track bugs so a bug tracking system is essential, don't develop with out some tracking mechanism. Metrics are easily available when you track defects and you can tell the health of your projects by looking at bug find rates.

Documentation

Documentation can make or break your product with users. Involve the documentation folks early – as early as prototyping, spec writing, and requirements.

If you want to maximize speed, don't print documentation – make it web-based and you avoid long delays in getting things printed

The End Game

The project has gone very well, you've shipped – are you done yet? No, there are still things to set up for the next release and these include:

- Post-mortem
- Final bug resolution
- Celebration

10 Lessons Learned

1. Pay Attention – projects don't run themselves – Active Project management
2. Review the work of new people, particularly if they are inexperienced. Avoid surprises
3. Do not skimp on analysis unless you are willing to rework things again and again.
4. Give people space to explore and play, but make sure that first things come first: the project.
5. If something is not working, let it go. And the sooner you let it go, the better.
6. It is important to engage everybody's mind on the project. Don't try to do all the thinking.
7. Having documentation enables you to bring in more people to help out.
8. Know the capabilities of the people you are working with (strengths and weaknesses)
9. Every task in the project plan should be associated with a deliverable and every deliverable should have a task that it is assigned to, no matter how trivial.
10. The earlier you can start testing, the better.

Bibliography

1001 Ways to Energize Employees by Bob Nelson

1001 Ways to Reward Employees by Bob Nelson

Black-Box Testing : Techniques for Functional Testing of Software and Systems by Boris Beizer

Built to Last : Successful Habits of Visionary Companies James C. Collins, Jerry I. Porras

Code Complete : A Practical Handbook of Software Construction by Steve McConnell

Death March : The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects by Edward Yourdon

Developing User Interfaces : Ensuring Usability Through Product & Process by Deborah Hix, H. Rex Hartson

Dynamics of Software Development by Jim McCarthy

Effective Methods for Software Testing by William Perry

Make It So : Leadership Lessons from Star Trek : The Next Generation by Wess Roberts, Bill Ross

Object Oriented Software Testing : A Hierarchical Approach by Shel Siegel, Robert J. Muller

Principle-Centered Leadership by Stephen R. Covey

Rapid Development : Taming Wild Software Schedules by Steve McConnell

Rapid Software Development With Smalltalk (Advances in ObjectTechnology Series, 7) by Mark Lorenz

Seven Habits of Highly Effective People by Stephen R. Covey

Smalltalk by Example : The Developer's Guide by Alec Sharp

Software Testing in the Real World : Improving the Process by Edward Kit, Susannah Finzi (Editor)

Software Testing with Visual Test 4.0 by Thomas R. Arnold II

Testing Computer Software by Cem Kaner, Jack Falk, Hung Quoc Nguyen

The Art and Science of Smalltalk (Hewlett-Packard Professional Books) by Simon Lewis

The Craft of Software Testing : Subsystem Testing Including Object-Based and Object-Oriented Testing by Brian Marick

The Death of Competition : Leadership and Strategy in the Age of Business Ecosystems by James F. Moore

The Fifth Discipline : The Art and Practice of the Learning Organization by Peter M. Senge

Working With Difficult People (Worksmart) by William, Phd Lundin, Kathleen Lundin

URLS:

Change, designing fool proof software, innovation:
http://www.fastcompany.com/08/change.html

Too Much to Do, Too Little Time: http://www.fastcompany.com/04/time.html

Tips For Time Shifting http://www.fastcompany.com/04/time2.html

Self-Help for the Super-Stressed http://www.fastcompany.com/06/selfhelp.html

How to Disagree (Without Being Disagreeable):
http://www.fastcompany.com/01/disagree.html

Linda McInnis

Linda McInnis is an associate director for quality engineering resources which encompasses responsibility for quality assurance, technical documentation, release engineering, data operations, and Y2K. She has 20 years of industry experience in these areas both as a consultant, individual, and managerial contributor. She is the author of several software development process models and many articles on how to develop software, test, and deploy it.

Ms. McInnis holds a B.S. in physics from Worcester Polytechnic Institute and has pursued graduate study in electrical engineering at Stanford University. She currently is listed in *Who's Who in the East* and *Who's Who of Business Leaders* and is a member of the ACM and IEEE societies.

Marco Ocaña

Marco Ocaña is software development project leader for Millennium Pharmaceuticals, Inc. He has been instrumental in the process improvement and process model development efforts at Millennium. He is currently an active project manager of multiple successful development projects.

Mr. Ocaña holds a B.S. in industrial engineering from Worcester Polytechnic Institute and is pursuing his M.S. in computer science from Worcester Polytechnic Institute. He is a member of the ACM and IEEE societies.