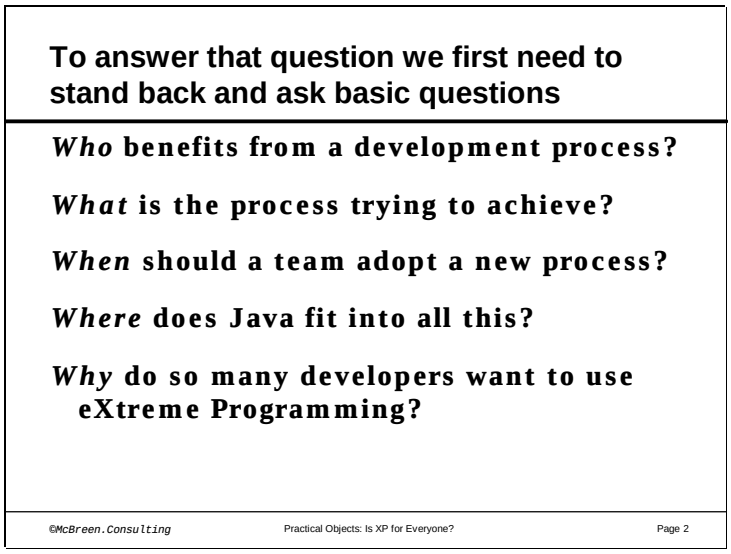
 © 2000 McBreen.Consulting

Is XP for Everyone?

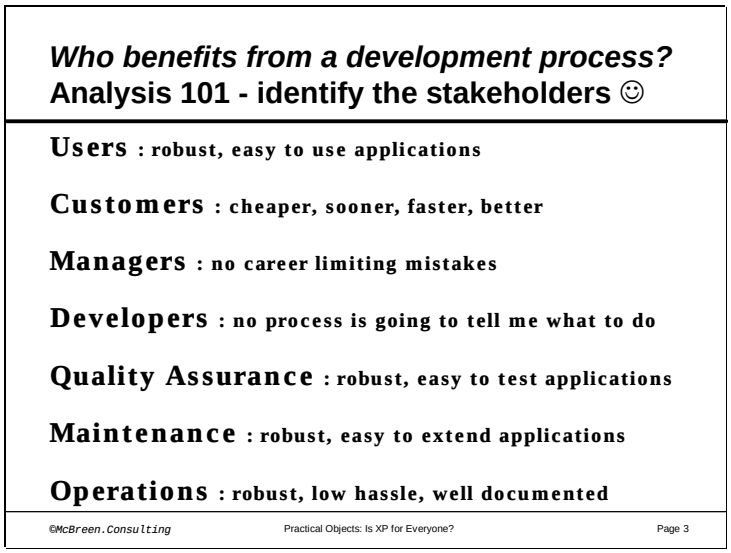
eXtreme Programming (XP) is a new development process that is starting to dominate the conversation about lightweight methodologies. As developers we need to remember that XP is not the only game in town. There are lots of different ways of obtaining similar results including SCRUM, Adaptive Software Development and the Crystal Methodologies. What is important in all of this is that a development team choose an appropriate process to suit the needs of their organization and project. Topics that will be covered include

- Exploring the methodology space
- The purpose of a development process
- Impacts and implications of XP and other lightweight processes
- How Java has changed development processes



This talk covers these 5 basic points

Alistair Cockburn talks about the Methodology Space as a way of describing processes.
Impact of Errors vs. Team size being a good starting point, but many more dimensions to talk about.



This is the really hard question that few people really address. Processes rarely exist for the benefit of developers, they are there for Managers and Customers. When selecting a new process this is the audience to which you have to speak.

Developers do have an independent streak, so we have to allow for this when choosing a process.

Maintenance and operations are often overlooked, but they are integral to the long term success of any system.

Notice how many of the stakeholders really value a robust application - everyone who has to deal with the system on a day to day basis.

© 2000 McBreen.Consulting

What is the process trying to achieve?
It depends ☺ Look at the process goals

eXtreme Programming : simple, high quality

Adaptive Software Development : Internet RAD

SCRUM : creating successful products quickly

FDD : frequent, tangible working results

Crystal Clear : human-powered software development

Unified Process : one-size-fits-all tailorable process

OK, so I'm being slightly facetious about the unified process, it really should read "Sell more CASE tools"

- XP - software development is simple, don't get bogged down building tools to build applications, build applications instead
- ASD - build adaptable systems that can evolve in Internet time while still being robust, mission critical applications.
- SCRUM - expert friendly process

When should a team adopt a new process?
When your current process isn't working ☺

Switching processes is very hard because it means you have to change your habits

Processes are cultural and behavioral

- Organizational and team values change
- Roles change within the team
- The activities that people do are different
- The artifacts and deliverables are different

Adopting a different development process will take a long, long time

All teams have a process, rarely is it a standard process.

Need to re-emphasize that changing processes is exceedingly difficult

Where does Java fit into all this?
Java requires Object Oriented Development

OO development is different, programmers write classes and methods not programs

OO developers use classes written by other people to add features to applications

OO design requires the identification of appropriate abstractions

The value of writing program specifications for 3 line methods is questionable

Java has been the thing that has allowed OO development to come into the mainstream of software development.

Prior to Java OO was a niche thing, now it's hot. Unfortunately few teams have much experience using it so they are suckers for any process that claims to be good for OO development.

OO development fails miserably when traditional development processes are used

Analyst, Designer, Architect, Programmer and Tester are not good roles in OO

A designer can implement a class in less time than it would take to fully specify all the methods

Programmers have to know the entire class library to avoid creating near duplicate functionality

OO needs *developers* who can contribute to all of the activities in the process

Without this, incremental development and evolutionary delivery are practically impossible

©McBreen.Consulting Practical Objects: Is XP for Everyone? Page 7

Very, very easy to crash and burn an OO project. Few people have been there, done that, got the T shirt.

Dark side of objects is that teams can get seduced by the idea of building a framework to help in the delivery of the application. *This used to be the Smalltalk problem, but Java has inherited this behavior.*
Analysts need to code so that they understand what is really possible.
Designers and architects need to talk to the users and actually implement something so that they know that their grand plans actually are workable and are useful for their users. Testers have to be involved with the users to discover exactly what the requirements mean. Testers also design and code testing frameworks.
Specialization might be useful and effective on really large teams, but I find it to be of dubious value on teams of 10 people and smaller.

Why do so many developers want to use eXtreme Programming?

XP addresses the balance of power between developers, managers and users

XP makes the customer pay for any requested documentation

XP says that developers can write really good, error free code

XP values and validates programming

XP is low stress and enjoyable

©McBreen.Consulting Practical Objects: Is XP for Everyone? Page 8

These are the nice answers, many more answers are possible.

Developers say how long a feature will take, users say how important a feature is.

Customers pay for documentation by accepting less functionality or a longer timeline.

XP promotes and values Refactoring to improve the design of the code, it says that we can write good code quickly, provided we have the appropriate practices in place. (XP deletes all invalid, low value comments)

XP is low stress, 40 hour weeks and Pair Programming help in this, as does having a complete automated test suite.

But eXtreme Programming is only safe to use if a project meets the preconditions

Users/Customers speak with a single voice and are available to answer all questions

All requirements can be easily expressed in a testable form that can be automated

Customers, managers and developers are willing to abide by Planning Game rules

Developers and managers are committed to use Pair Programming

These preconditions are a show stopper for many projects.

Getting answers from users fast requires them to be co-located and able to provide 10-20% of the person days on the project.

Testable requirements means that *XP is well suited to creating replacement systems*, since can use the existing system as a test Oracle.

Abiding by the planning game rules can be very hard for some organizations that are used to managing by putting pressure on developers to work faster.

Getting commitment to Pair Programming is not always easy, facilities may need changing to encourage pairing, developers have to be comfortable working intensely with each other.

If a project does not meet the preconditions don't even bother trying XP

Remember, XP is not the only way to play the software development game

- The software engineering approach using hordes of average programmers is not always optimal
- It might be a lot better to start paying attention to our skills in the craft of software development
- ***Software Craftsmanship*** suggests that small teams of good developers are the way forward

Software Development is meant to be fun, if it isn't the process is wrong

XP can be very fast and productive, but only if the conditions are right. If many competing stakeholders need to be consulted before making any decision on requirements is an obvious show stopper. XP uses user stories that are a promise of a future conversation, not an explicit requirement contract.

But as we saw on slide 4, many processes to choose from, each being optimized for different aspects and situations.

One of the reasons XP is so successful is that it creates a good learning environment to enable developers to get really good fast. But the practices that enable this are easy to replicate within other processes - short design sessions before coding, writing the unit tests first (Test Driven Development), refactoring and pair programming all facilitate communication between developers about what is good design and development.

Personally, I think the time has come to start questioning the hordes of average programmers approach of software engineering and instead focus on what we can do to nurture the development of outstanding developers so that we can create even large systems with small teams.

Software Craftsmanship is my label for this, book should be out Q3, 2001.