Extreme Programming for Solo Projects

Dr. Peter Lappo
peter.lappo@smr.co.uk
www.smr.co.uk

Agenda

- Why Methods?
- What is Extreme Programming (XP)
- XP Risk Management
- Characteristics of Solo Projects
- Applying XP to Solo Projects
- Case Study
- Conclusion and Questions

Why Methods

- There is uncertainty (risk) in
  - Requirements
  - Schedule or resources
  - Development
  - External factors
- Lack of discipline
  - Scope and requirements creep
  - Gold plating
- Others .....

Risk Management

- In essence software and project management methods are simply risk management tools.
- A set of practices and tools to help the project deliver business value within economic constraints.

What is Extreme Programming

- XP consists of:
  - four values.
  - *approx* 12 practices that complement the values.
    - different “gurus” have different practices!
    - cool or confusing?

XP Values

- Feedback
- Communication
- Simplicity
- Courage

XP Practices

- The Planning Game
- Small Releases
- Design Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- Sustainable Pace
- On-site Customer
- Coding Standards

ref. Kent Beck

XP Risk Management

- How does XP manage risk?

Short Iterations

- Typically 2 weeks long.
- Customer defines what to deliver.
- Easy to change focus at each iteration.
- Deliver code not documents.
- Release cycle 3-4 months.

- Customer is in control.
- Customer always gets something useful.

Customer Involvement

- Customer and developer dialogue during planning.
- Customer and developer dialogue during development.

- Software meets customers needs.
- Customer is in control.

Self Motivated Teams

- Teams work together on,
  - Estimating and task breakdown.
  - Design and coding.
- Software developed in pairs – but not always.
- Daily stand-up meetings to review progress.

- Two heads are better than one.
- Less reliance on "key" personnel or experts.

Automated Testing

- Automated tests at unit level.
- Tests at acceptance test level to prove requirements.
- Tests become the detailed requirements.
- Test first design.

- Tests are the critical enabler of change.
- Impact assessment easier.

Software is Easy to Change

- Refactoring simplifies code and improves design.
- Test first design forces better structure.
- Simple design.
- Collective ownership of code.

- Makes software easier to change and understand.
- No “no go” areas.

Light on Documentation

- Focus on delivering working code.
- Only produce documentation that adds value,
  - Such as overview documents.

- Reduces workload.
- Deliver real business value.

Feedback and Control

- Customer decides what goes in each iteration.
- Dialogue during iteration.
- Collect stats on performance.

- Fine degree of customer control.
- Estimates improve with time.

XP Risk Management

- XP reduces uncertainty in
  - Requirements
  - Schedule or resources
  - Development
  - External factors
- XP is disciplined
  - Reduces scope and requirements creep
  - Reduces gold plating

Benefits

- Valuable code in production early.
- Easier for the customer to change their minds.
- Robust set of tests for the entire life cycle.
- Accurate view of project status.
- Closure on features.

Characteristics of Solo Projects

- Solo projects will have varying degrees of
  - customer involvement
  - other developer involvement
- Other factors tend to remain constant.
- Two types
  - customer driven
  - self motivated (no customer)

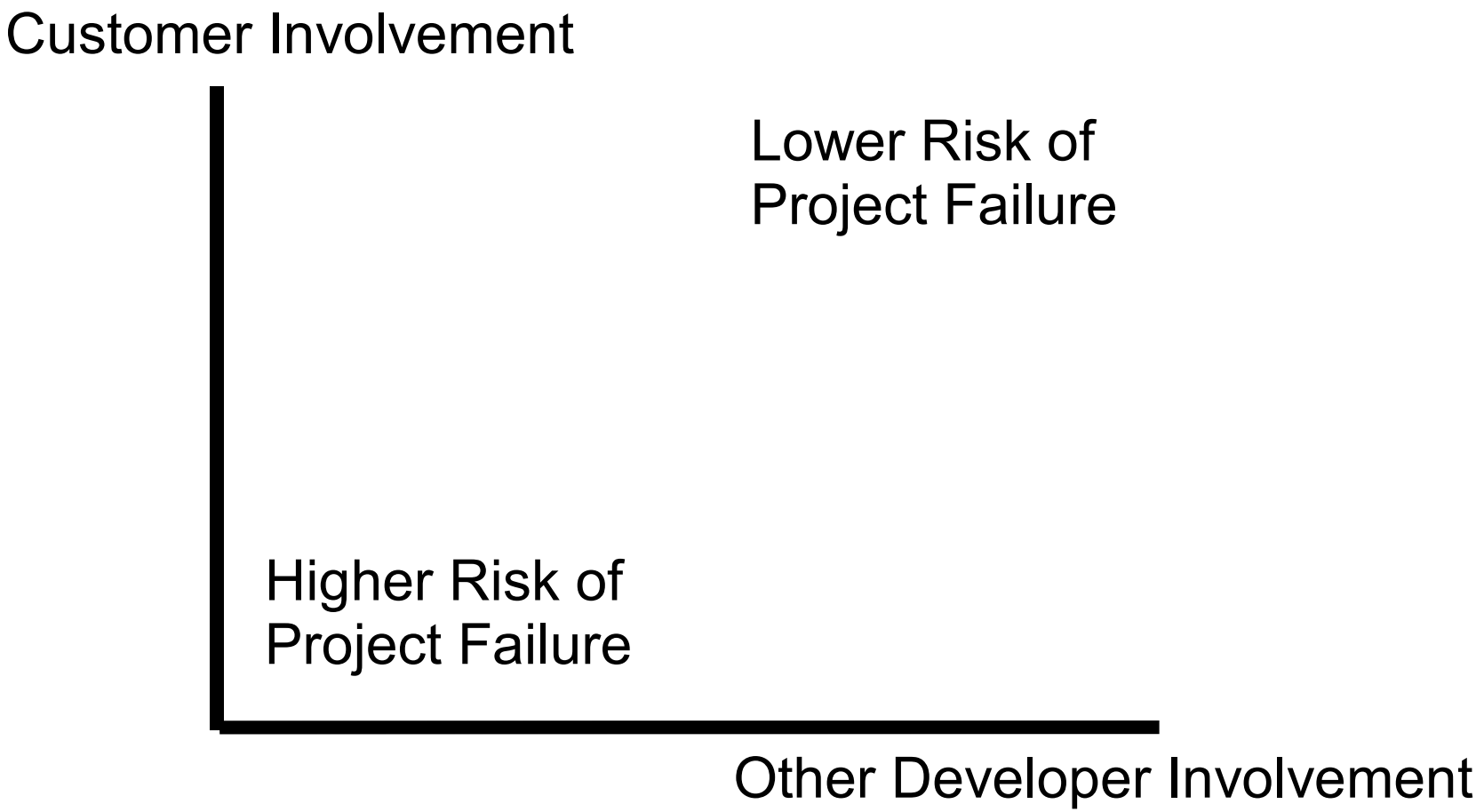
Customer Involvement

- Customers may be
  - deeply involved, completely absent, or somewhere in between.
- As customer involvement tends to zero so,
  - Feedback and communication reduces.
  - Ownership of user stories passes to the developer and is less likely to be correct.
  - Harder to assess progress.

Developer Involvement

- Other developers may be involved indirectly
  - occasional pair programming with colleagues
  - peer review
  - support and help via Internet or work place.
- As developer involvement tends to zero, so
  - fewer unit tests
  - discipline looser
  - design could be less understandable

Characteristics of Solo Projects



Applying XP to Solo Projects

- Solo projects need a customer
  - developer must play the role of the customer
  - must be critical from a “customer” perspective in terms of requirements, usability etc.
  - no problems scheduling customer meetings !!
- Solo projects need developer help
  - don't stop learning
  - sign up to discussion groups
  - find someone to peer review

XP Show Stoppers

- XP or agile software development is not easy to use on any project.
- Enhancing a system using XP is not easy especially if there are no automated tests.
- Not recommended for safety critical systems.
- Not easy to apply to fixed price projects but it can be done.

Requirements or User Stories

- User stories (aka use cases) are used to capture requirements in XP.
- XP doesn't have a lot to say on the process of creating user stories.
- If you have a customer specification break it up into stories. It will be more manageable and easier to measure progress.
- If you don't have a specification brain storm the stories over an hour or two. Chances are you'll capture most of them.

User Stories (more)

- User stories shouldn't be take longer than five days to implement at the most.
- Break up big stories into smaller ones. Measuring progress and testing become easier.
- User stories are usually written by customers, but if you do write your own stories get them reviewed by a tutor, friend or colleague.

Analysis and Design

- XP is not a great fan up big up front analysis and design as models often turn out to wrong in practice.
- However, some analysis and design is useful to get a better understanding of the problem and determine the overall shape of software.
- Further analysis and design is done as part of each story implementation.

XP Practices for Soloists

- The Planning Game
- Small Releases
- Design Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- Sustainable Pace
- On-site Customer
- Coding Standards

ref. Kent Beck

The Planning Game

- The planning game allows the customer to define the business value of desired features, and uses cost estimates provided by the programmers, to choose what needs to be done and what needs to be deferred.
- Stories are used to define XP projects.
- The planning game is run at the start of each iteration.
- The effect of planning game is to steer the project to success.

The Planning Game (more)

- Without a customer steering is much harder.
- The planning game can be faked by asking your customer or yourself to prioritise the user stories.
- Always ask yourself what “business value” the feature is providing.
- **Don't** include anything that is not adding value. e.g. its all too easy to fiddle with a screen design to make it look “nice”.

Small Releases

- XP teams put a simple system into testing as soon as possible, and update it frequently on a very short cycle.
- Iterations are typically 2-3 weeks.
- Releases are typically 2-3 months.
- Iterations should be viewed as mini software projects from analysis to test and delivery.

- This can easily be done by a soloist.

Design Metaphor

- XP teams use a common "system of names" and a common system description that guides development and communication.
- There may be more than one metaphor.

- This can easily be done by a soloist as it will help in maintenance.

Simple Design

- The delivered program should be the simplest program that meets the current requirements.
- There is no building "for the future".
- Instead the focus is on providing business value.
- Good design is simple design.
- Complex designs are harder to understand and riskier to change.

Simple Design (more)

- Of course it is necessary to have good software design.
- In XP design happens during the planning game, when the story is being implemented and post implementation through "refactoring".

- Soloists have to be especially careful in creating simple designs as no one is reviewing their efforts.

Testing

- XP teams focus on validating the software.
- Programmers develop software by writing a test first then coding to make the test pass.
- Unit tests should be automated.
- Expect at least 50% of your code to be unit tests.

- Soloists can easily write automated unit tests as these are developer tests.

Testing (more)

- Customers provide acceptance tests that enable them to prove their desired features exist.
- Customer acceptance tests are more difficult for a soloist especially if there is no customer or the customer cannot write them.
- Automate customer acceptance tests if you can.

Refactoring

- XP teams improve the design of the system throughout the entire project.
- This is done by keeping the software clean: without duplication, with high communication, simple, yet complete.

- A very powerful design technique but you really need unit tests to prove your software still works.

Pair Programming

- XP programmers write production code in pairs, two programmers working together at one machine.
- Pair programming has been shown to produce better software than programmers working alone, but costs can be an issue.

Pair Programming (more)

- Soloists can't pair programme, unless perhaps:-
  - you ask someone to discuss a software problem
  - commit your code to open source
  - post to a discussion group
  - people are always willing to help

Collective Ownership

- All the code belongs to all the programmers.
- This lets the team go at full speed, because when something needs changing, it can be changed without delay.

- Not applicable to a Soloist.
- You own all the code!

Continuous Integration

- XP teams integrate and build the software many times per day.
- This keeps all the programmers on the same page and enables more rapid progress.

- Not applicable to a Soloist.
- You are the only person working on the software!
- But don't forget to use a version control system.

Sustainable Pace

- Tired people make more mistakes.
- Do not over work unless you really have to meet a deadline (like a presentation).
- Better still ensure essential stories are developed at the beginning and defer non-essential stories to the end.
- Drop non-essential stories if you run out of time.

On-Site Customer

- An XP project is steered by a dedicated individual who is empowered to determine requirements, set priorities, and answer questions as the programmers have them.
- The effect of the customer being there is that communication improves, with less hard-copy documentation - often one of the most expensive parts of a software project.

On-Site Customer (more)

- This is a big issue for soloists as communication is one of the keys to successful software implementation.
- If you don't have a customer try and fake a customer as best you can.
- There is a risk your software won't be as good without the adequate feedback so try and release to users as soon as practical.

Coding Standards

- For a team to work effectively in pairs, and to share ownership of all the code, all the programmers need to write the code in a consistent way, with rules that make sure the code communicates clearly.
- Even though this is a group practice its still good practice to have coding standards.
- If your software is successful someone else will read it one day, so follow industry norms and don't invent your own standards.

Case Study
Trading Gateway Development

- System to route and translate trade messages to and from a stock market.
- “Real-time” and written in Java.
- A gateway feeding orders and trades to an order manager from a sales system.

Case Study - Development of a Share Trading Gateway

- Practices used – without anyone realising!
  - The Planning Game
  - Small Releases
  - Design Metaphor
  - Simple Design
  - Testing
  - Refactoring
  - Pair Programming
  - Collective Ownership
  - Continuous Integration
  - Sustainable Pace
  - On-site Customer
  - Coding Standards

Conclusion

- Using XP practices is possible on solo projects but its not strictly XP. But do you care?
- Being an XP soloist is just like being a soloist in a band. To get good results it is
  - hard work
  - requires discipline
  - needs lots of practice

Summary and Questions

- A **lot of value** can be gained on a solo project from:-
  - short iterations (small releases)
  - simple design
  - refactoring
  - automated testing
  - talking to others

The End