Trinadex

Agile and XP

for
Non-traditional Development

by
Stephen McHenry
Managing Principal

Trinadex Corporation
www.trinadex.com

Agile – Core Principles

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

  Individuals and interactions over **processes and tools**
  Working software over **comprehensive documentation**
  Customer collaboration over **contract negotiation**
  Responding to change over **following a plan**

- That is, while there is value in the items on the right, we value the items on the left more.

Principles behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

Principles behind the Agile Manifesto (cont'd)

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Principles behind the Agile Manifesto (cont'd)

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile
Sweet Spots

- Two to eight people in one room
- Onsite usage experts
- One-month increments
- Fully automated regression tests
- Experienced developers

Extreme Programming (XP)

- Small to medium sized teams developing software in the face of vague or rapidly changing requirements.

- 2-10 People
- Rapidly Changing Requirements

Extreme Programming (XP)
The Practices

- The Planning Game – Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.
- Small Releases – Put a simple system into production quickly, then release new versions on a very short cycle.
- Metaphor – Guide all development with a simple shared story of how the whole system works.
- Simple Design – The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

Extreme Programming (XP)
The Practices (cont'd)

- Testing – Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating what features are finished.
- Refactoring – Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify or add flexibility.
- Pair Programming – All production code is written with two programmers at one machine.
- Collective Ownership – Anyone can change the code anywhere in the system at any time.

Extreme Programming (XP)
The Practices (cont'd)

- Continuous Integration – Integrate and build the system many times a day, every time a task is completed.
- 40-Hour Week – Work no more than 40 hours a week, as a rule. Never work overtime a second week in a row.
- On-site Customer – Include a real, live user on the team, available full-time to answer questions.
- Coding Standards – Programmers write all code in accordance with rules emphasizing communication through the code.

Moments in XP History

- Chrysler C3 project
- Earlier project
  - Payroll system replacement project – Y2K driven
  - 1995 Timeframe
  - Generated GUI screens
  - Bad tax calculations
- Rework of earlier project
  - 1996 – Kent Beck + Ron Jeffries
  - Throw away and start over
  - Two years in, lots of hype
  - Feb 2000 – project cancelled with no follow-on phase

On- Site Customer

- Quote

  “Once you accept that scope is variable, then suddenly the project is no longer about getting it ‘done’. Rather, it’s about developing at a certain velocity. And, once you establish a velocity, then the schedule becomes the customer’s problem.”

  Robert C. Martin

On- Site Customer (Original)

- Real experts required – full time
- Unavailable for two weeks (sick, vacation, etc.)
- Can't remember exactly (nothing written down until customer acceptance tests)
- Inconsistent – tells different things to different people / different interpretations
- Doesn't know everything and "fakes it" (pressure to keep the project moving – may make snap decisions)
- Single biggest point of failure

On- Site Customer (Original)

- Customer may become influenced by technical issues, rather than business issues
- Lost benefit of being at the customer site (and hearing the interactions)
- Too big a job for one person (complaint)

On- Site Customer (New)

- Customer teams equal or larger than development teams - KB
- Keeping the team in one room
- Budget issues with larger teams
- “One voice” likely lost

Pair Programming

- Social Dynamics
- Lack of Privacy
- Lack of “quiet thinking time” – noisy room
- Ergonomic Issues
- Cost justified?
  - ~Doubles the cost
  - ~Doubles the “finding people” problem

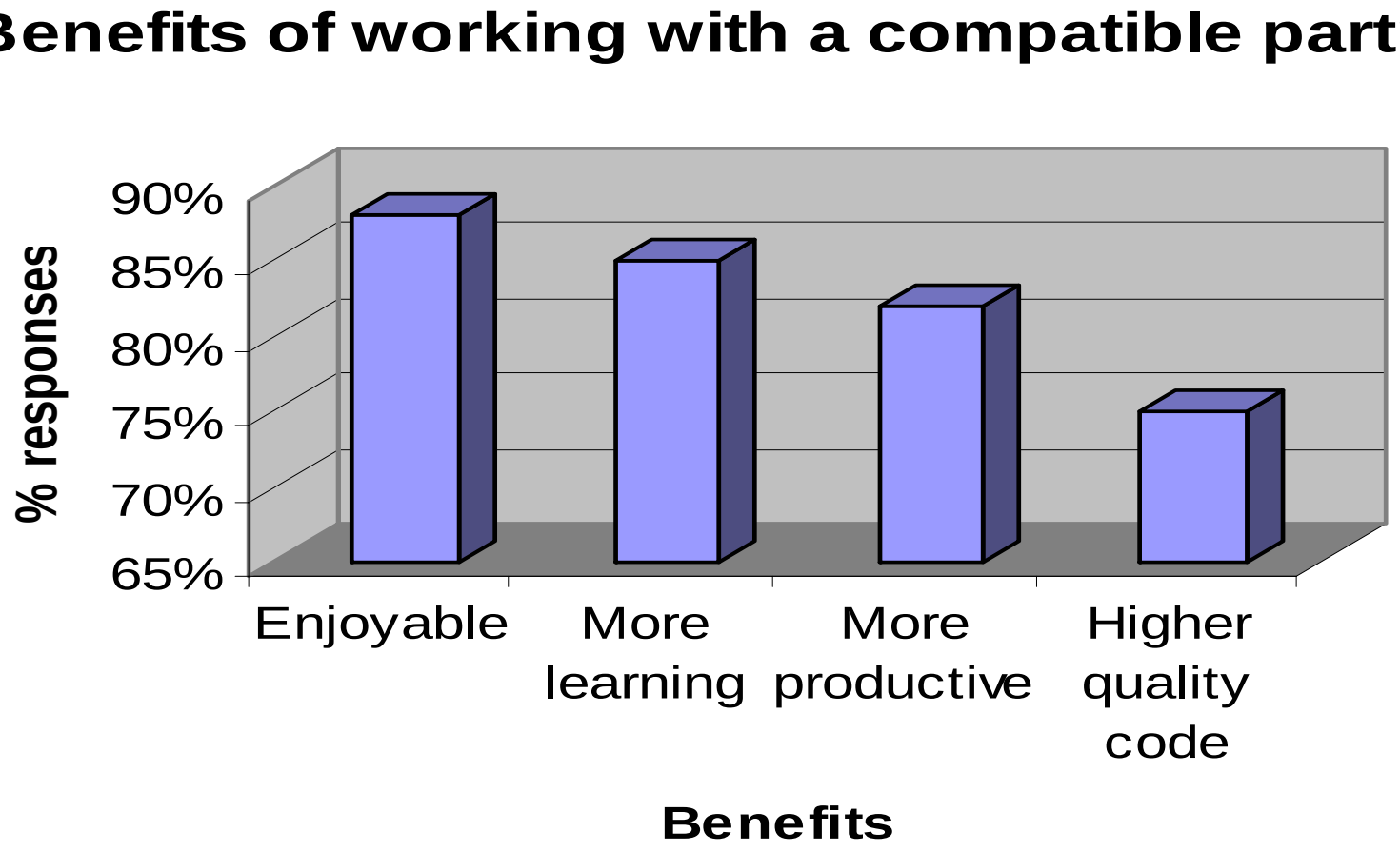
Pair Programming

- Different categories of programmer
  - Expert-expert
  - Expert-average
  - Expert-novice
  - Novice-novice
  - Extrovert-extrovert
  - Extrovert-introvert
  - Introvert-introvert
- Hogging the keyboard
- Everybody gets sick at once

Qualities of Most Compatible Partners

Work ethic	71% (240)	Same gender	27% (92)
Sense of humor	65% (221)	Project mgmt skills	25% (84)
Personality match	61% (207)	Punctual	22% (76)
Similar skill level	61% (206)	Different gender	21% (73)
Felt comfortable	56% (191)	Lower skill level	20% (69)
Work patterns	44% (150)	Similar age	15% (51)
Work participator	40% (135)	Same ethnicity	7% (25)
Higher skill level	33% (111)	Same nationality	6% (21)

Benefits of Most Compatible Partners

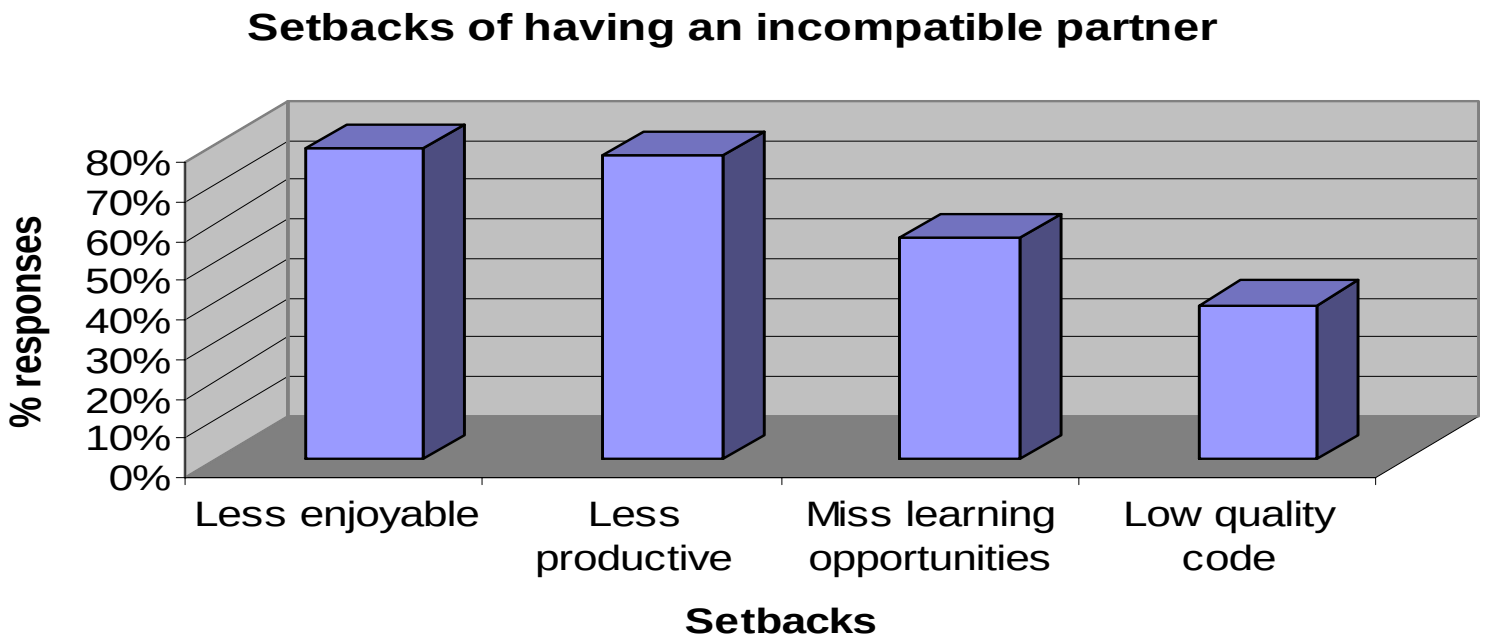


Trinadex

Qualities of Least Compatible Partners

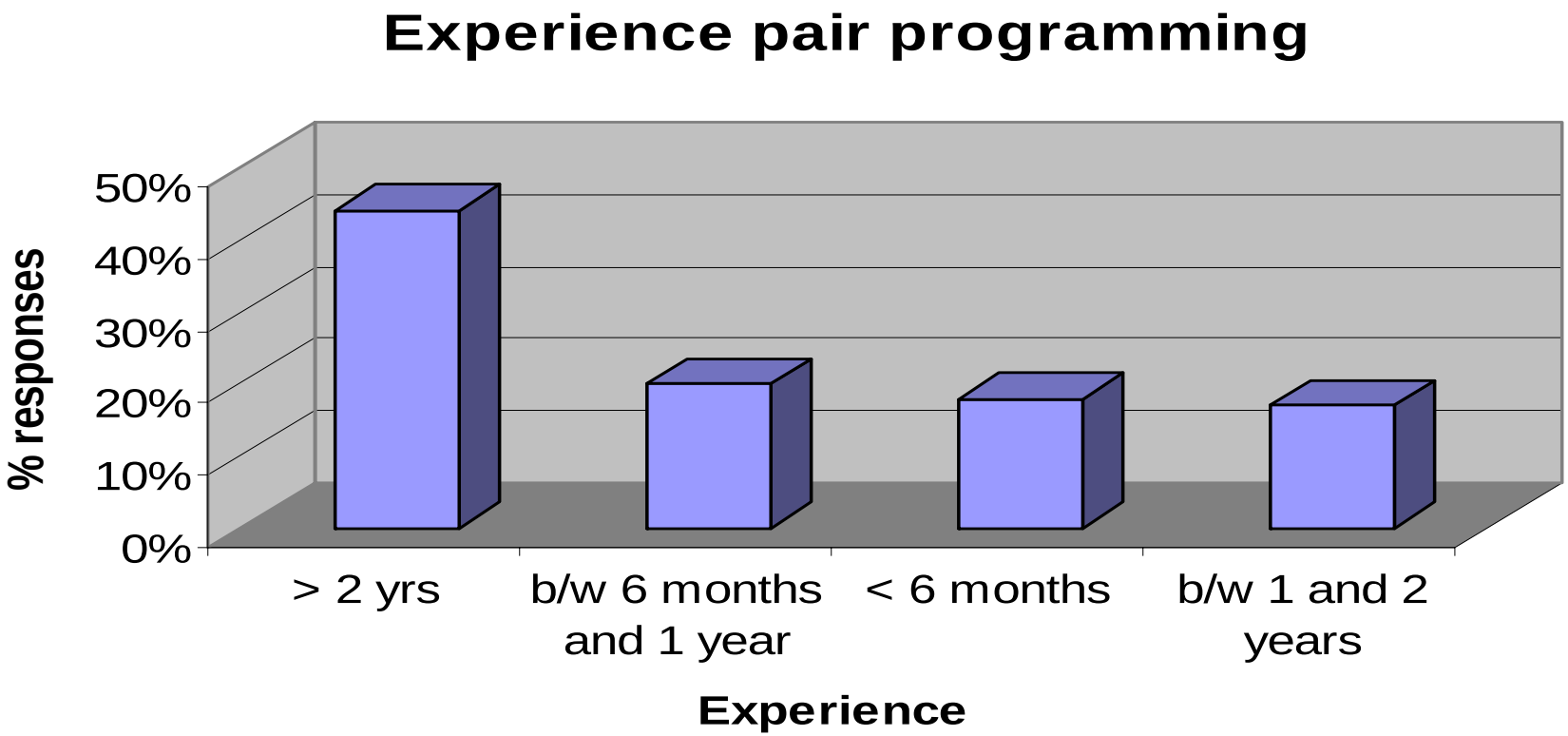
Personality mismatch	55% (182)	Made you inferior	16% (52)
Diff's in work ethic	44% (146)	Language problems	15% (50)
Not participatory	44% (145)	Not punctual	13% (42)
Lower skill level	32% (108)	Ask personal questions	7% (22)
Sense of humor	28% (93)	Higher skill level	6% (21)
Did not talk enough	27% (91)	Different gender	5% (17)
Breath problems	20% (65)	Same gender	4% (13)
Body odor	19% (64)	Similar skill level	4% (13)
Different work patterns	18% (61)	Age differences	3% (9)
Different PM skills	17% (58)	Different nationality	2% (8)
Too talkative	17% (57)	Different ethnic bkgd	2% (6)

Setbacks of Least Compatible Partners



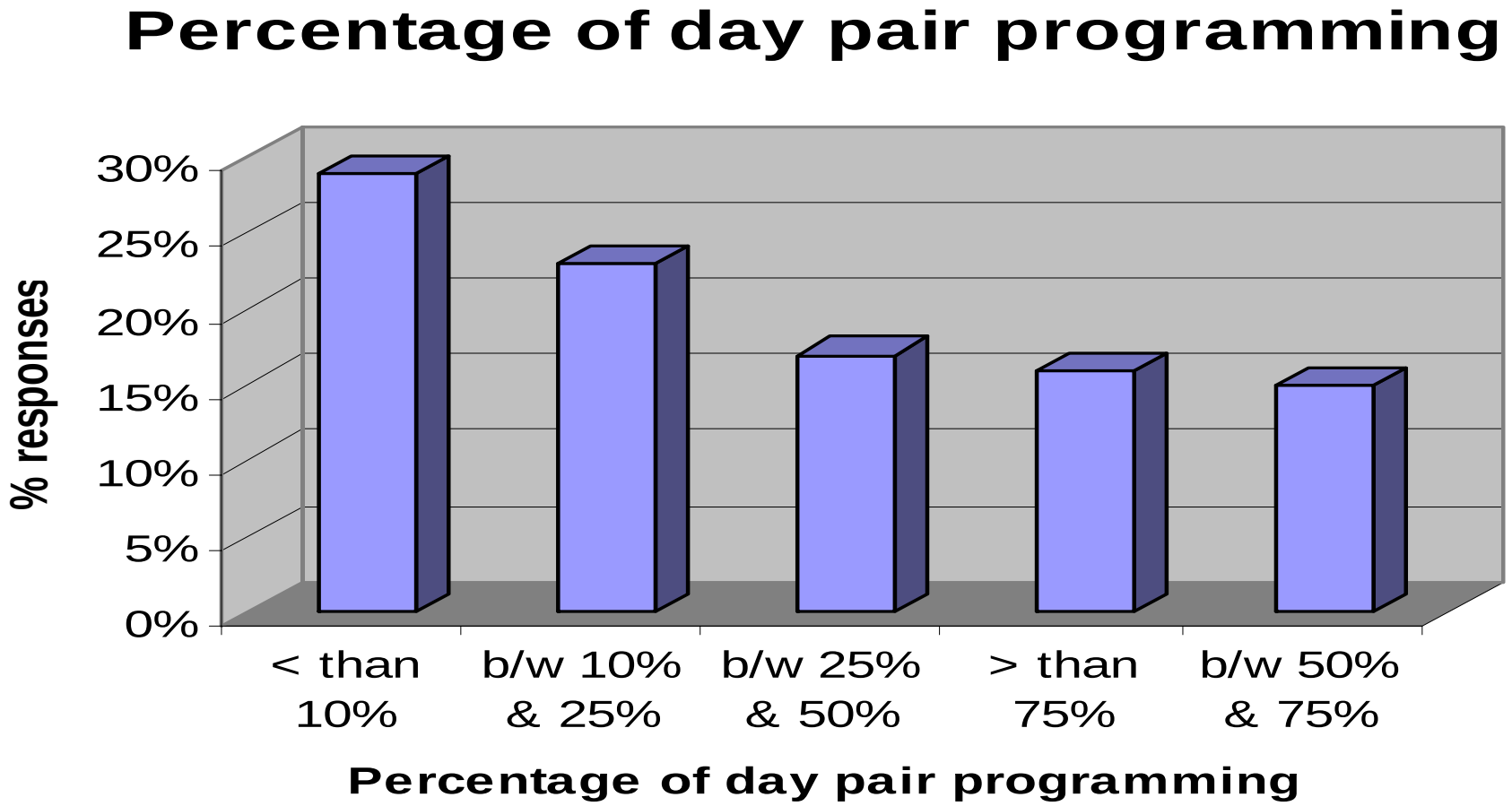
Trinadex

Experience with Pair Programming



Trinadex

Percentage of Day Pair Programming



Trinadex

Enjoy Pair Programming and Why

Yes 91% (304)	Learn 77% (257) Higher quality code 73% (244) More productive 71% (236) Social aspect 57% (190) Avoid long debugging sessions 49% (156)
No 9% (29)	Like working alone 5% (16) Get more work done alone 5% (15) Feel like I'm teaching my partner all the time 4% (14) Produce high quality code on my own 3% (11) No sense of accomplishment 2% (8) Don't get along with partner 0% (0)

Oral Documentation

- One or two sentence “user stories” captured on story cards – “promises” of future conversations
- Conversations during iteration
- Documentation
  - Not prohibited, but not encouraged
  - Not under change control
  - Code is the documentation
- Programmers that get hired midstream
- People
  - Forget
  - Change their mind

Unit Testing

- Only catches anticipated bugs
- XP Programmers write their own tests → errors of omission
- Rigorous adherence to testing practices could result in more test code than system code
- Bugs in unit tests
- Not all code can be unit tested
  - Asynchronous Messaging
  - Multithreaded systems

Constant Refactoring

- Refactoring IS useful
  - Tool for improving the design
  - Not as substitute for design
- Wasted work
  - Prevented by upfront requirements followed by design
  - Time to refactor, no time to write down requirements
- Often requires “guerilla tactics”
- Knowing when to stop (smell the code)
- Annoying the users
  - *Refactoring the UI*
  - *Refactoring Live Data*
  - *Corrupting the Database*

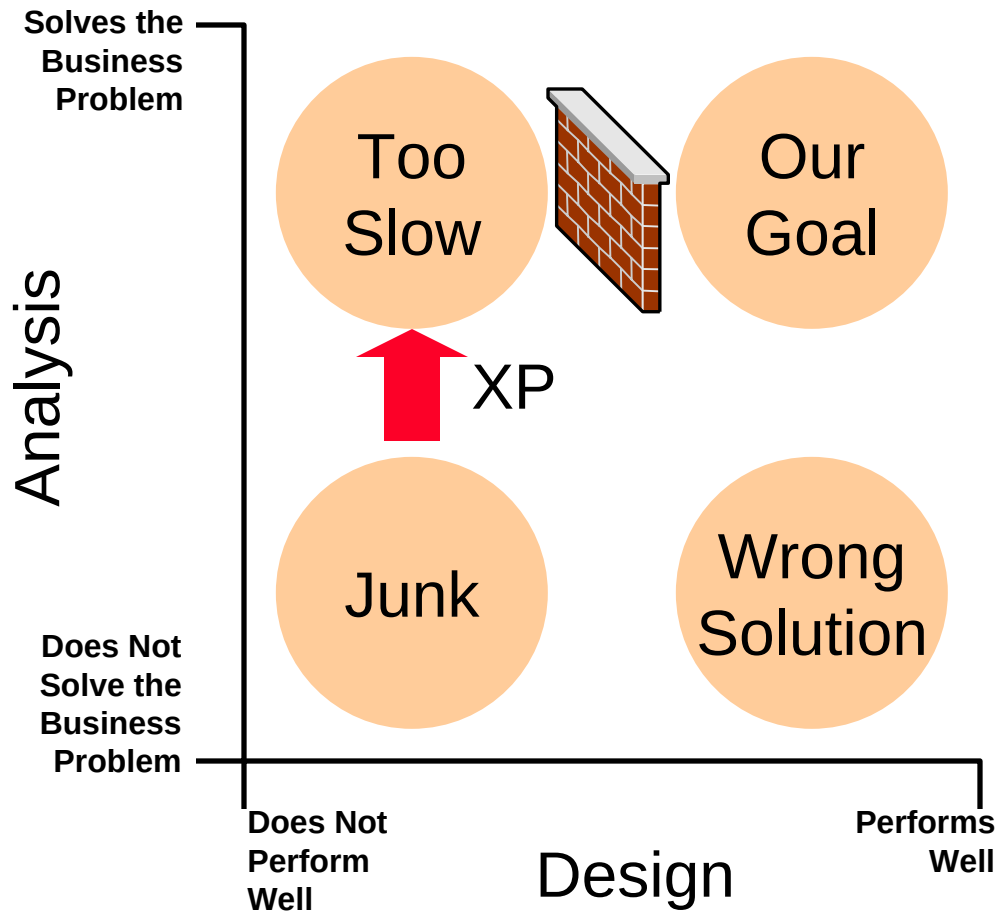
Constant Refactoring

- Refactoring is NOT inexpensive
- Time consuming
- Stopping criteria not well defined
- Premature code release – all maintenance, all the time
- “If it ain’t broke, don’t fix it” – still good advice
- Refactoring databases problematic – especially 24/7
- Refactoring UI on live systems - problematic

Emergent Design

- YAGNI – You Aren't Going to Need It
- Frameworks (for design)
- Problem areas – Orthogonal to functionality
  - Scalability
  - Multiple platforms
- Problems
  - Lack of overall design clarity
  - No "gestation" period before coding
  - Paper design easier/faster to change than code
  - Early definition of interfaces allows parallel development
  - Lack of "big picture" for impact analysis

Analysis vs. Design



Trinadex

Emergent Design

- Emergent design (substitutes for planning ahead)
  - Payroll
  - Operating Systems
  - Telephone Switch
  - EFT
  - LASIK Beam Control Software
  - Autopilot
  - NORAD
  - Space Station Environmental Control
  - Missile Guidance
  - Air Traffic Control

Other Problem Areas

- Dates are hard dates, but scope varies – Optional Scope Contract

XP Cycle

- No detailed written requirements
  - Used on risky projects
  - “Dynamic” requirements are handled by →
- Emergent Design
  - No upfront design – handled by →
- Constant Refactoring
  - Required due to “make up as you go” philosophy
  - Could cause lots of bugs, but those are caught by →
- Unit Testing
  - Good for coding errors, but design errors require human intervention
  - Human component supplied by →

XP Cycle

- Pair Programming
  - Help each other with design and coding issues
  - Rotated frequently to increase code familiarity
  - Reduced accountability is solved by →
- Collective Ownership
  - No one responsible / Everyone responsible
  - Constant refactoring by different pairs could pull code in opposite directions
  - No spec to arbitrate
  - System could stray from customer desires. Solved by →

XP Cycle

- On-Site Customer
  - Junior customer (Real decision maker for a year? Not likely)
  - Role is inherently challenging and stressful → turnover likely
  - Continuity of “customer” – an issue
  - Problem - Requirements in their head
  - Solved by →
- No Detailed Written Requirements

Trinadex

www.trinadex.com