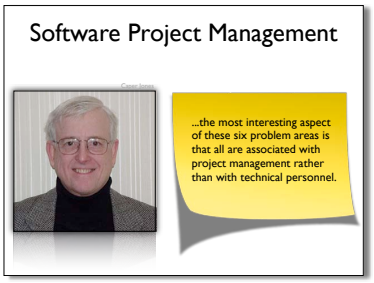
Agile Software Development

Rahul Premraj + Andreas Zeller
Saarland University

Credits: Rory V O'Connor

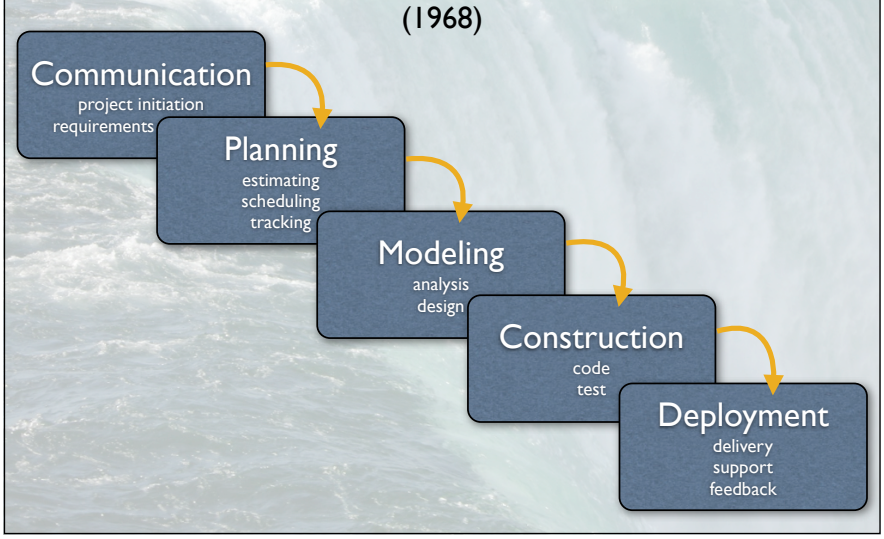
Reasons Projects Fail



- Lack of end-user (customer) involvement
- Poor requirements
- Unrealistic schedules
- Lack of change management
- Lack of testing
- Inflexible and bloated processes

Waterfall Model

(1968)



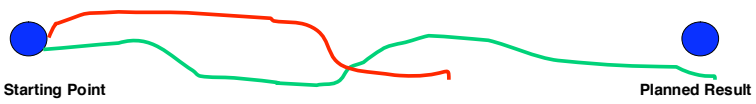
4

Waterfall Risks

- Relies heavily on accurate assessment of requirements from the very start.
- Customers only see product at the end of the life-cycle. Little or no interaction or feedback earlier on.
- Problems found later on are much harder to correct.

5

Code and Fix Model

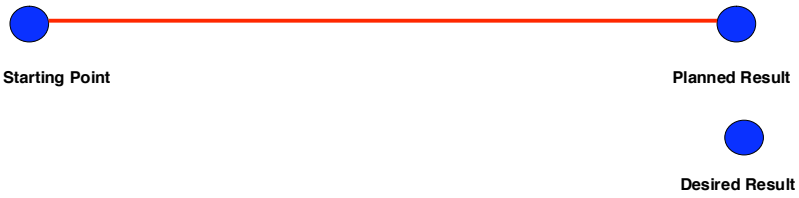


"Fire, fire, fire, fire, aim again, fire, fire, fire--there is no time for 'ready'."

-- Gary Hamel, *Leading the Revolution*

6

Heavy Weight Disciplined Model



Ready, ready, ready, ready, aim, aim, aim... fire.

7

Adaptive, Lightweight Agile Processes

Ready, aim, fire, ready, aim, fire, ... fire.

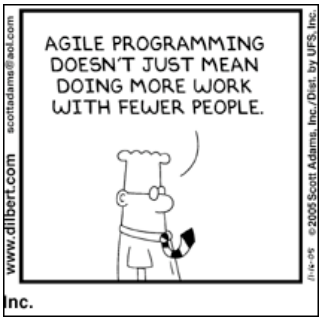
8

If a traditional process is like a battleship, protected against everything that might happen…

9

an agile process is like a speedboat, being able to change direction very quickly

What is Agile Development?



Inc.

Agile Alliance

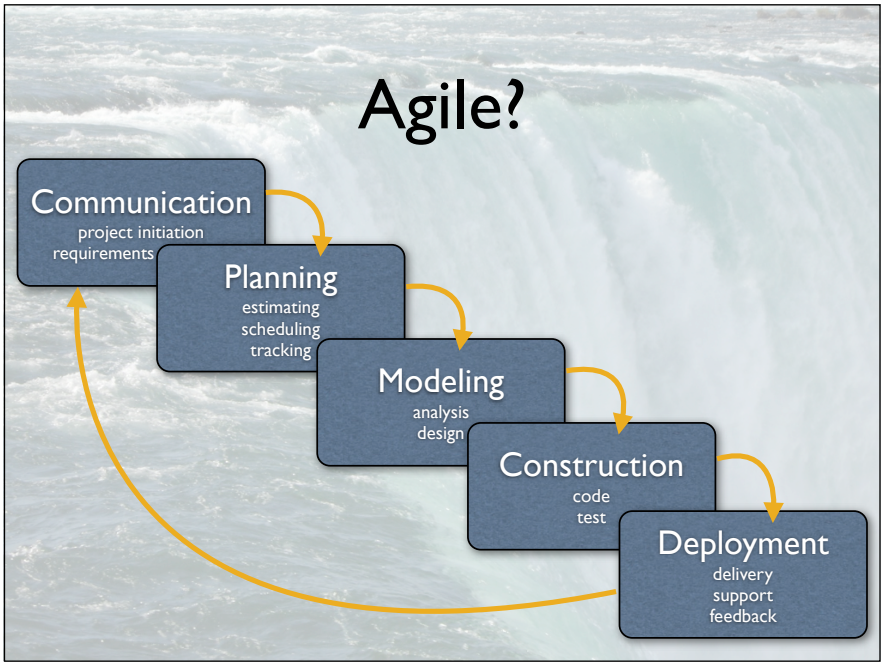
Manifesto for Agile Software Development (2001)

- Individuals and activities over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan..

What is Agile Development?

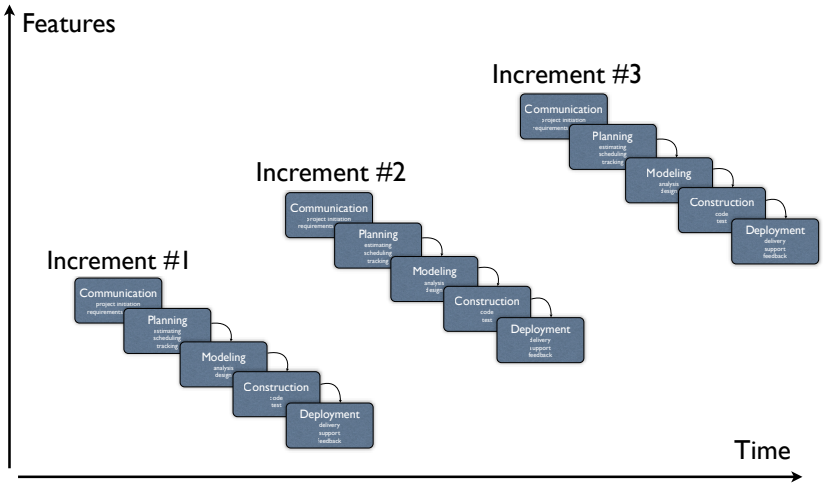
- Fast development? Hacking? Prototyping? Uncontrolled fun? Programmer heaven?
- Agility = ability to react to changing situations quickly, appropriately, and effectively.
  - notice changes early
  - initiate action promptly
  - create a feasible and effective alternative plan quickly
  - reorient work and resources quickly and effectively

13



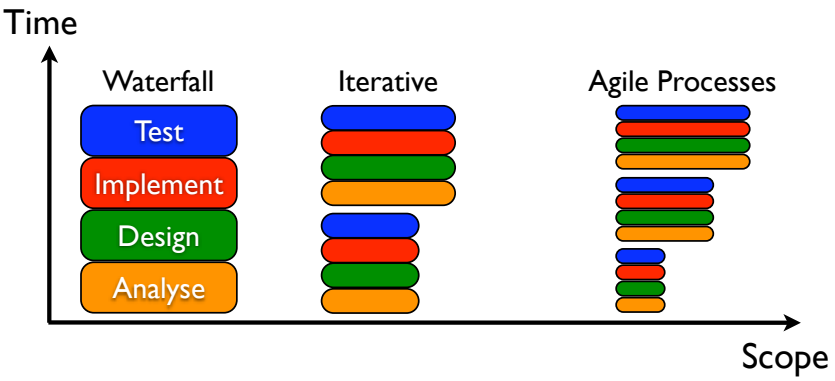
14

Incremental Model



15

Agile Processes



Credits: Prof. Bodik

Agile vs. Plan-driven

Agile	Plan-driven
• Low criticality	• High criticality
• Senior developers	• Junior developers
• Requirements change very often	• Requirements don't change too often
• Small number of developers	• Large number of developers
• Culture that thrives on chaos	• Culture that demands order

What is an Agile Process?

- Difficult to predict which requirements will persist or change in the future.
- For many types of software, design and development are interleaved.
- Analysis, design, construction, and testing are not as predictable.

So, how to tackle unpredictability?

make the process adaptable...

but...

Kitchen Sink Syndrome

Adapt Incrementally

- An agile team requires *customer feedback*.
- Effective *catalyst* – operational prototype or portion of the system.
- Deliver *increments in short times* so adaptation keeps pace with change.
- Iterative approach allows customer to *evaluate product* regularly.

Human Factors

- Competence
- Common focus
- Collaboration
- Decision-making ability
- Fuzzy-problem solving
- Mutual trust and respect
- Self-organisation

Agile Process Models

A number of agile process models exist.

There are many similarities (in philosophy and practice) among these approaches.

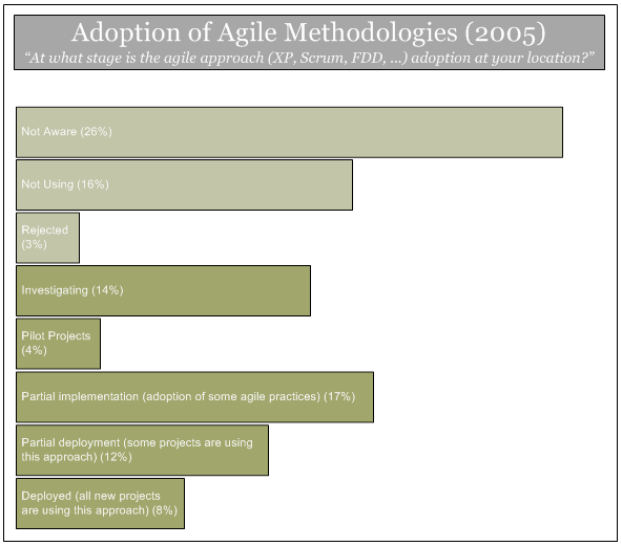
Agile Process Models

A number of agile process models exist.

There are many similarities (in philosophy and practice) among these approaches.

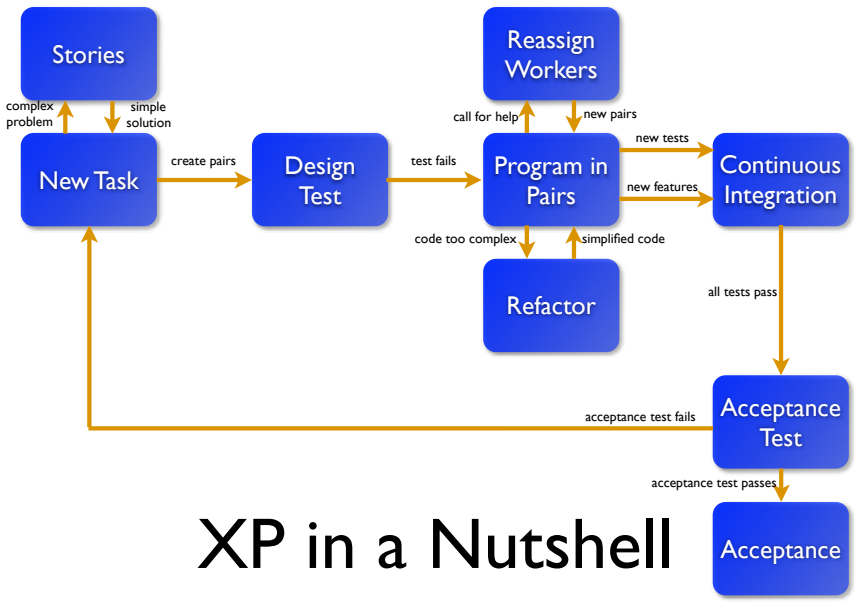
- Adaptive to requirements changes . . . embracing change
- Iterative/Explorative
- Process itself adapts to project . . . disciplined, yet flexible
- Rely on collaboration/innovation through group interaction
- Simplicity… "Just do it."
- Minimal documentation (efficiency, low inertia)

Agile Methods Adoption

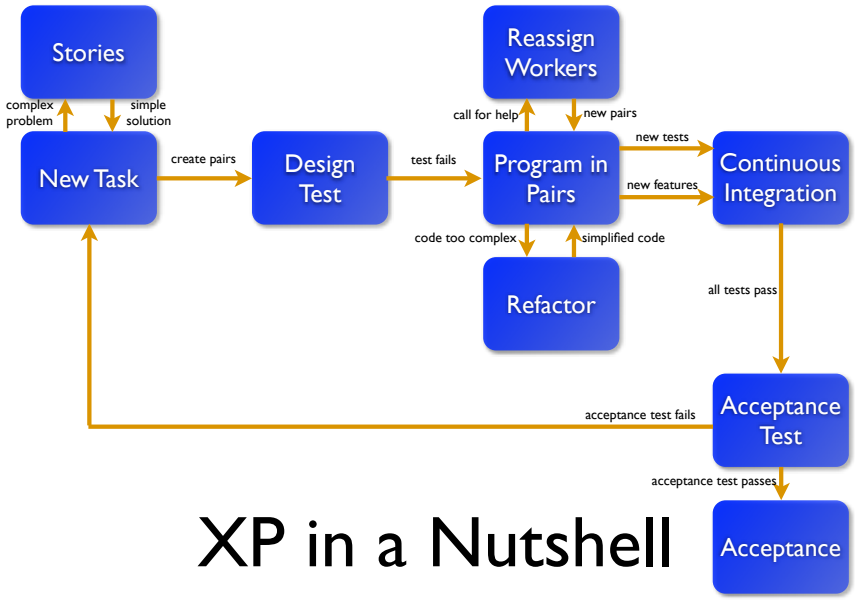


Extreme Programming (XP)

- Based on ideas and methods from the late 1980s.
- Published in 1999 by Kent Beck.
- Doesn't adhere to any particular language, but generally more suited to object oriented development.



XP in a Nutshell



XP in a Nutshell

XP: Planning

- Creation of *user stories* that describe features and functionality for software to be built.
- Stories are written by the customer on an index card.
- Customers assign a priority or value to the card and developers assign a cost.

XP: Planning

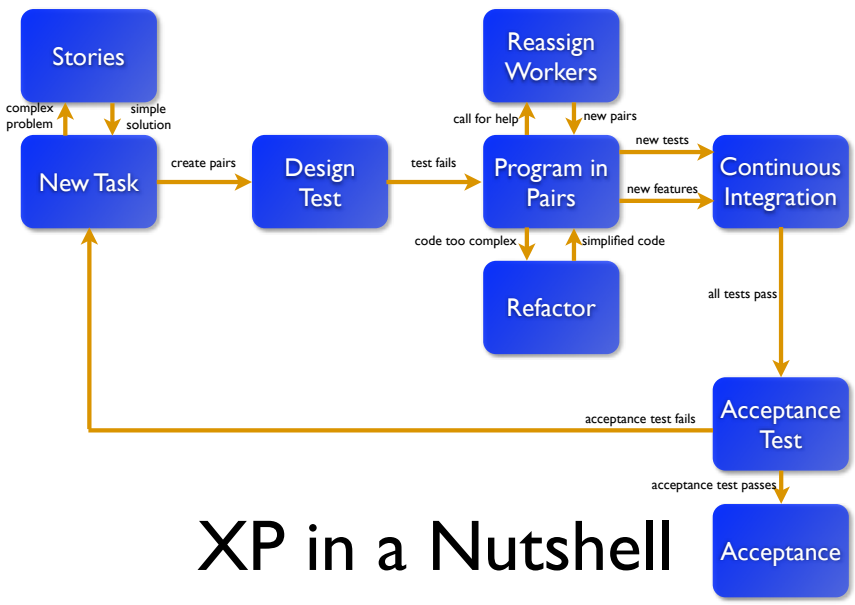
- If a story takes longer than 3 development weeks, the customer is asked to split the story into *smaller stories* and cost and value are assigned again.
- Both sides agree on which stories are to be grouped for next release.

XP: Planning

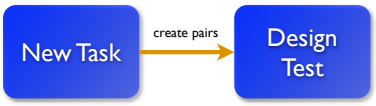
- XP team orders implementation by
  - all stories implemented immediately.
  - stories with highest priority implemented first.
  - stories with highest risk implemented first.

XP: Project Velocity

- *Project velocity:* Number of customer stories implemented in first release
- helps estimate delivery rates and schedule for further releases.
- determine whether an over-commitment has been made for all stories across the entire project.

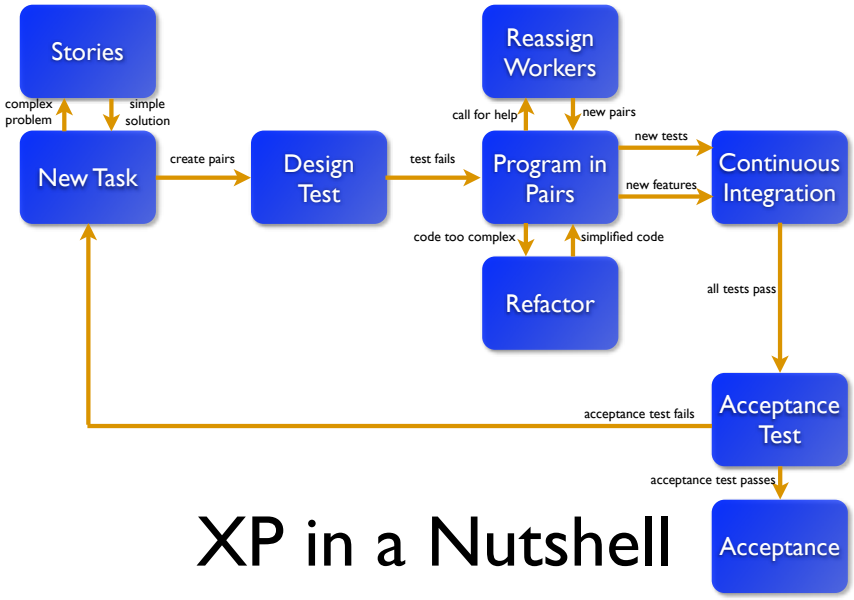


XP in a Nutshell

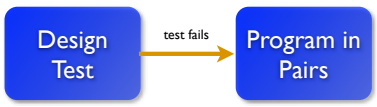


XP: Test first

- *Test first* – Test cases are written before code
- Initially, tests must fail (no code yet)
- Replaces (formal) specification
- Validated by executing the test



XP in a Nutshell



Pair Programming

- Coding is traditionally split amongst the development team with each taking a particular function or specification and generating the code and unit tests.
- In XP all code is built by two programmers, sitting side by side, at the same machine.

Pair Programming

- Pair programming ensures that all production code is reviewed by at least one other programmer, and results in better design, better testing, and better code.
- Knowledge transfer amongst the team is also a major advantage when using pair programming.
- Both programmers in the pair are familiar with the code and have either written the code or has been actively involved as the programmer watching the code generation.

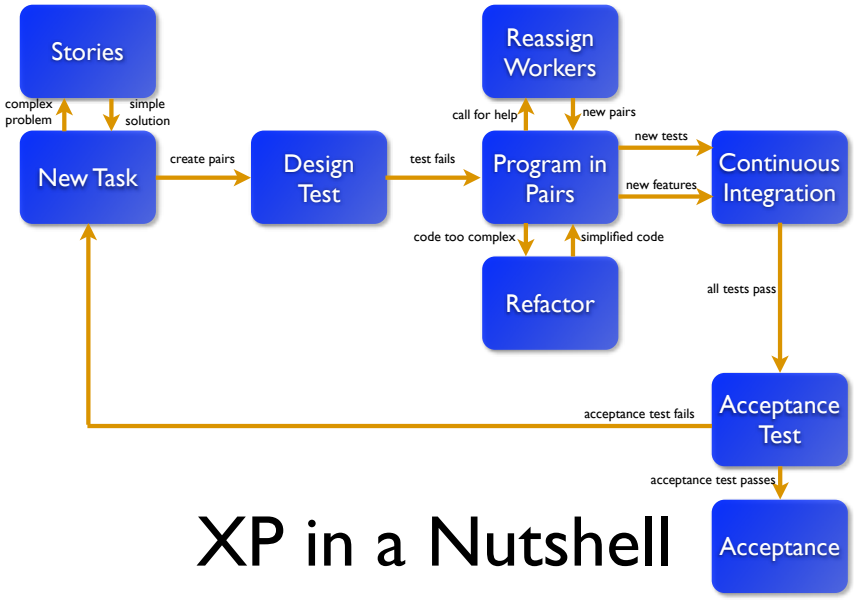
Some Rules

- Specialisation leads to queues (you will block on the specialist at some point).
- The 'Driver' implements, focusing on the tactical.
- The navigator is more objective (strategic): asks 'why' and provides explanations.

Pair Programming

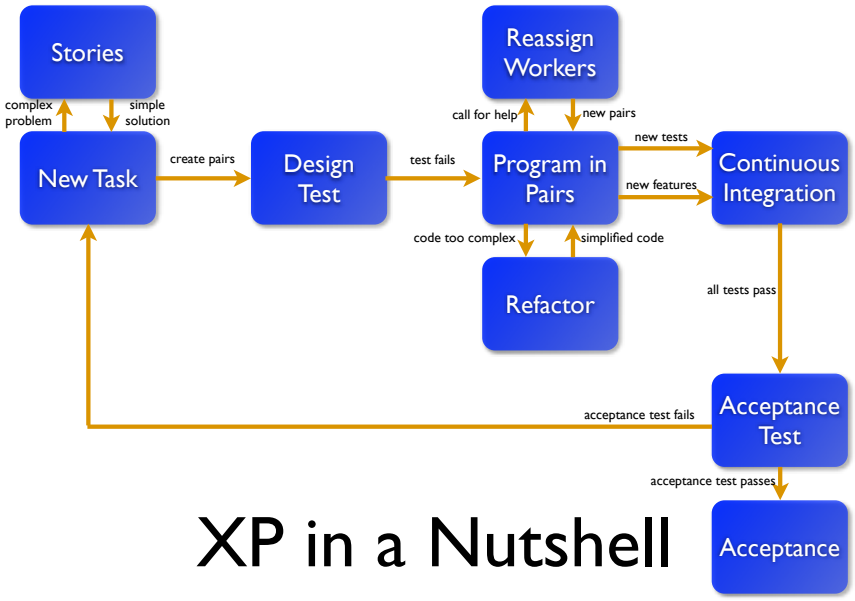
Some Rules

- Healthy pairings have communications every 45 to 60 seconds.
- Pairs should switch roles often.
- Pairs can break off for low complexity tasks.
- Pairing is intense (suggest a break every 2 hours).



XP in a Nutshell

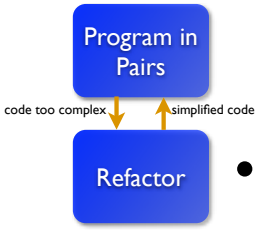
Reassigning Pairs



XP in a Nutshell

43

http://en.wikipedia.org/wiki/Refactoring



Refactoring

- *Refactoring:* any change to a computer program's code which improves its readability or simplifies its structure without changing its results.
- Polish existing code to make it better.
- Improve or change design, architecture, efficiency, quality
- *but not* functionality

44

XP: Design

- Basic Principle: "KIS" (Keep It Simple)
- Simple design always preferred over a more complex representation.
- Design provides story implementation guidance.
- Extra functionality is discouraged.

45

XP: Design

- Encourages use of CRC (class-responsibility-collaborator) cards – see lecture on Software Architecture.
- Only design work product in XP process.
- Difficult design problems are tackled by an immediate prototype (spike solution) and evaluation. This is to lower risk in the future.

XP: Coding

- Coding does not follow stories and initial design.
- *Unit test cases* are developed to help better focus on what has to be implemented.
- Once coding is complete, tests are run to get instantaneous feedback.

Refactoring

- Not something to dedicate two weeks for every six months.
- Do as you develop.
- Refactor when you recognize a warning sign (a "bad smell") and know what to do.
  - … when you add a function
    - Likely it's not an island unto itself
  - … when you fix a bug
    - Is the bug symptomatic of a design flaw?
  - … when you do a code review
    - A good excuse to re-evaluate your designs, share opinions.

Duplicated Code

- Same expression in two methods in the same class?
  - Make it a private ancillary routine and parameterise it
  - (Extract method)
- Same code in two related classes?
- Push commonalities into closest mutual ancestor and parametrize
  - Use template method DP for variation in sub-tasks
  - (Form template method)

Long Method

- Often a sign of:
  - Trying to do too many things
  - Poorly thought out abstractions and boundaries
- Best to think carefully about the major tasks and how they inter-relate. Be aggressive!
  - Break up into smaller private methods within the class
  - Delegate subtasks to subobjects that "know best"

Divergent Change

- Occurs when one class is commonly changed in different ways for different reasons.
- Likely, this class is trying to do too much and contains too many unrelated sub-parts.
- This is a sign of poor cohesion.
- Solution: Break it up, reshuffle, reconsider relationships and responsibilities.

Feature Envy

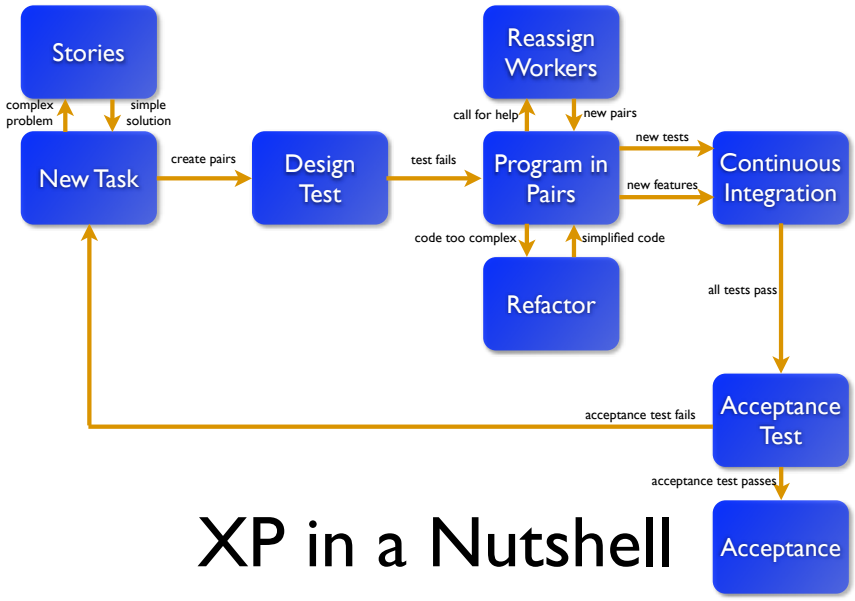
- A method seems more interested in another class than the one it's defined in
  - e.g., a method A::m() calls lots of get/set methods of class B
- Move m() (or part of it) into B!

Lazy Class

- Classes that doesn't do much that's different from other classes.
- If there are several sibling classes that don't exhibit polymorphic behavioural differences , the consider just collapsing them back into the parent and add some parameters.
- Often, lazy classes are legacies of ambitious design or a refactoring that gutted the class of interesting behaviour.

Refactor Mercilessly

- Improve the design of existing code without changing functionality
  - Simplify code
  - Opportunity for abstraction
  - Remove duplicate code
- Relies on testing to ensure nothing breaks in the process of refactoring.



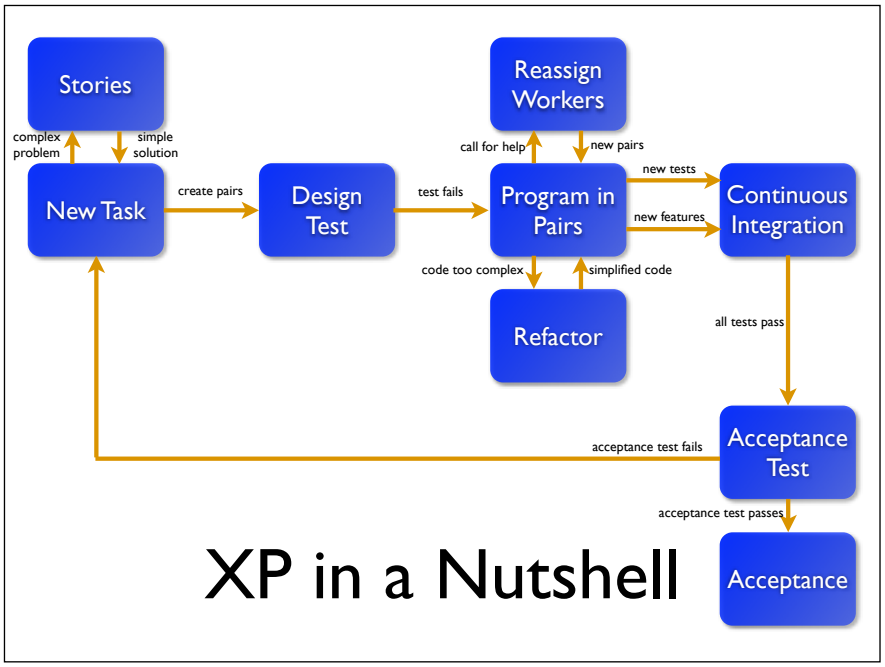
XP in a Nutshell

Continuous Integration

- When developers go to release new code, they run all the unit tests, not just theirs, on the integration machine.
- The tests must run at 100%.
- When a test fails, the problem must have been caused by the last change
- Reason: Developers know the tests ran at 100% the last time anything was released.

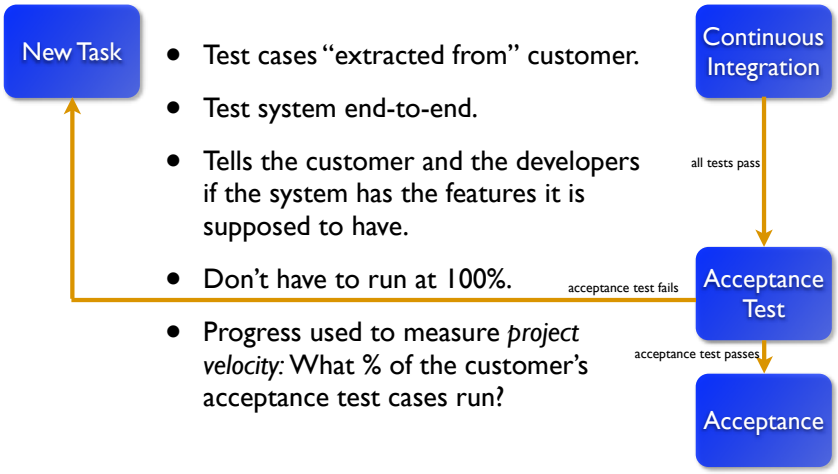
XP: Testing

- Implementation of unit tests should be such that they can be automated.
- As soon as code is modified, it can be tested.
- Integration and validation of the system can occur on a daily basis using universal test suite.
- A continual indication of progress.
- Acceptance tests are specified by customers to test features and functionality.



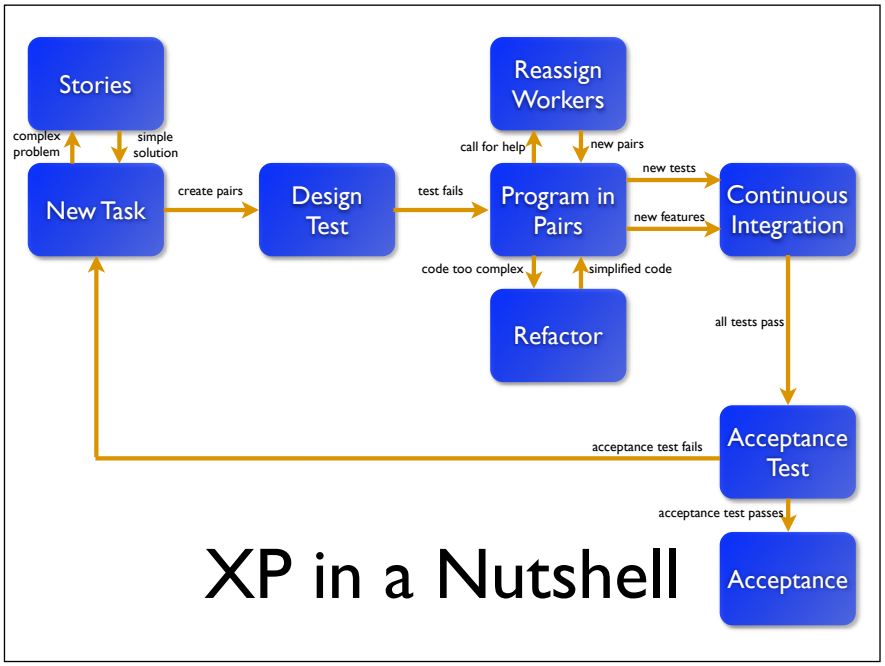
58

XP: Acceptance Tests



- Test cases "extracted from" customer.
- Test system end-to-end.
- Tells the customer and the developers if the system has the features it is supposed to have.
- Don't have to run at 100%.
- Progress used to measure *project velocity:* What % of the customer's acceptance test cases run?

59



60

Extreme Programming

XP: 12 Practices

1. The Planning Game
2. Small Releases
3. Metaphor
4. Simple Design
5. Testing
6. Refactoring
7. Pair Programming
8. Collective Ownership
9. Continuous Integration
10. 40-hour Week
11. On-site Customer
12. Coding Standards

1 The Planning Game

- Use stories to facilitate knowledge transfer
- Put decisions in the hands of the person with the best knowledge:

  business decisions ⟶ Customer

  software decisions ⟶ Developer
- Plan only as far as your knowledge allows (next iteration or next release)

2

Small Releases

- Supports quick feedback from users.
- Simplify the tracking of metrics.
- stories per iteration ⟶ project velocity
- Increase the manageability of the project for the customer.

3

Metaphor

- Ground all discussions in a single shared story of how the whole system works
- Provide an overarching view of the project
- Connect program to work process

4

Simple Design

- Design embodies only the needed complexity and no more.

  emphasis on top-down or bottom-up design as needed to meet this iteration's stories.

  extra complexity removed when discovered.

- Simpler designs are easier to modify, maintain, and describe. They decrease the cost of design changes, but give no notion of product line architecture.

5 Testing

6 Refactoring

7 Pair Programming

(We already know these)

8 Collective Ownership

- Code to belongs to the project, not to an individual engineer
- Anyone can change any code anywhere and no personal ownership of modules.
- Everyone is permitted access to all the code so everyone has a stake in knowing all of the code.
- As engineers develop required functionality, they may browse into and modify any class.
- They are responsible for keeping all the Unit Tests running (and writing new ones for new functionality). "You break it, you fix it."
- Requires deserved trust

9 Continuous Integration

- The system always works
  - there is always something to be released
- Similar to rapid releases
  - fast feedback to developers on problems
  - no 'big bang' integration disasters

10

40 Hour Week

- Knowledge can only be transferred at a limited rate
- Work for sustained speed, not a single sprint
- Burning the midnight oil kills performance
- If you mess with people's personal lives (by taking it over), in the long run the project will pay the consequences

11

On-site Customer

- A real, live user available full-time to answer questions as they occur.
- Programmers don't know everything.
- Business knowledge is the key to a successful business project.

12

Coding Standards

- Communication occurs through the code
- Common standard promotes understanding of other developers' code.
- Helps promote team focus.

XP: 12 Practices

1. The Planning Game
2. Small Releases
3. Metaphor
4. Simple Design
5. Testing
6. Refactoring
7. Pair Programming
8. Collective Ownership
9. Continuous Integration
10. 40-hour Week
11. On-site Customer
12. Coding Standards

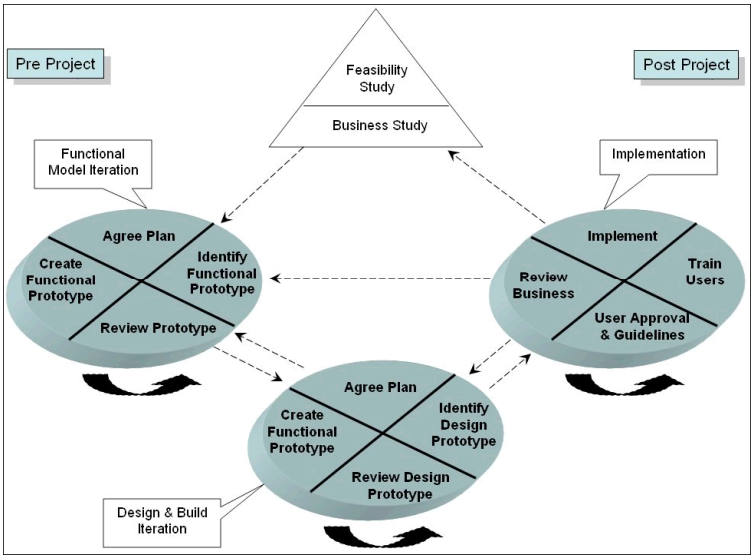
Adaptive Software Development (ASD)

- Process to build complex software and systems.
- Focuses on human collaboration and team self-organisation.

Dynamic Systems Development Method (DSDM)

- Suited for projects with tight time constraints.
- Use of incremental prototyping.
- Pareto Principle - 80% of the application can be delivered in 20% of the time it would take to deliver 100% of the application.
- The approach is applied to iterations. Only enough work is required to facilitate movement to the next increment. Details can be completed later.

DSDM Life Cycle



Scrum

Scrum

- Small working teams to maximize communication, minimize overhead and maximize knowledge sharing.
- Adaptable to technical and business changes.
- Yields frequent software increments that can be inspected.
- Development work and the people who perform it are partitioned into clean, low coupling partitions.
- Constant testing and documentation is performed.
- Ability to declare project "done" whenever required.

Scrum

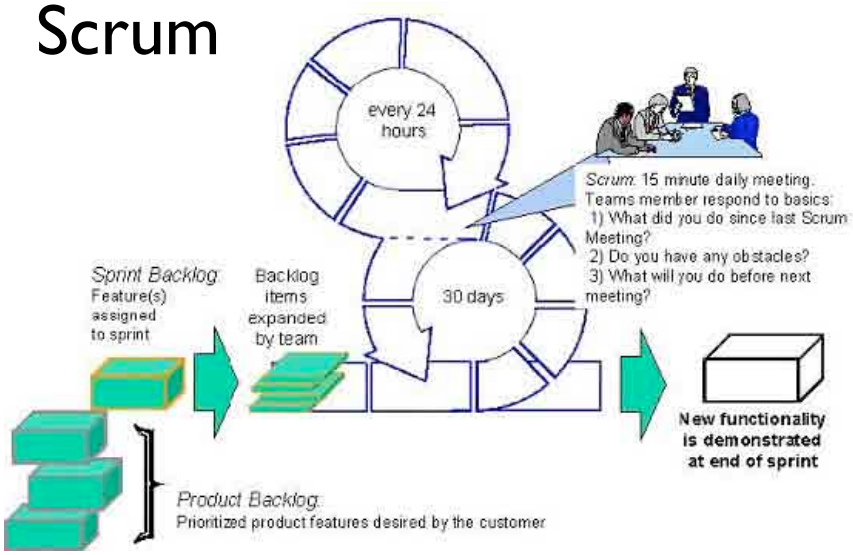
Backlog: A prioritised list project requirements or features that provide business value.

Sprints: Consists of work units that are required to achieve a defined backlog into a predefined time-box (usually 30 days).

Scrum Meetings: Short 15 mins. meetings held daily by the scrum team. The Scrum master leads the meeting.

Demos: Demonstrate software increment to the customer for evaluation.

Scrum



Feature Driven Development

- Initially conceived as a process model for object-oriented software engineering.
- Been extended and enhanced for agility and adaptation.

A feature is a client-valued function that can be implemented in two weeks or less.

Feature Driven Development

- Features are small blocks of deliverable functionality.
- Can be organised into a hierarchical business related grouping.
- Team develops operational features every two weeks.
- Design and code are easier to inspect.
- Project planning, scheduling, and tracking are driven by feature hierarchy.

Feature Driven Development

Template for defining a feature

<action> the <result> <by | for | of | to> a(n) <object>

Examples:

Add the product to a shopping cart.
Display the technical specifications of the product.
Store the shipping-information for a customer.

Feature Groups

<action><-ing> a(n) <object>

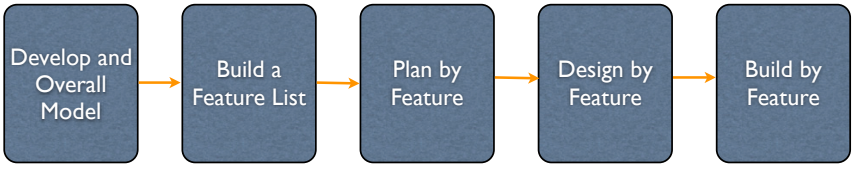
Making a product sale.

Feature Driven Development

more shape than content

A development plan (Class owners, feature set owners)

Completed client-value function



A list of features grouped into sets and subject areas

A design package (sequences)

Test-driven development

A software development technique consisting of short iterations where new test cases covering the desired improvement or new functionality are written first, then the production code necessary to pass the tests is implemented, and finally the software is refactored to accommodate changes.

http://en.wikipedia.org/wiki/Test_Driven_Development

Test-driven development

1. Add a test.
2. Run all tests and see the new one fail.
3. Write some code.
4. Run the automated tests and see them succeed.
5. Refactor code.

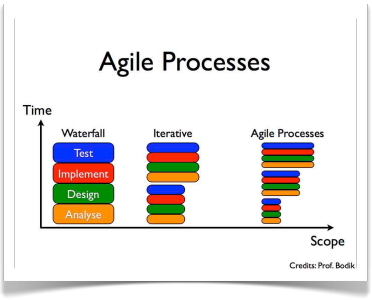
Agile Modelling

The scope and complexity of large systems should be modeled so that –

- all parties can better understand what needs to be built.
- the problem can be effectively split and allocated across people.
- quality can be constantly assessed at all stages of development.

Agile Modelling

- Model with a purpose: Have specific goals in mind when modelling (e.g. communicate some aspect with the customer).
- Travel Light: Different models may convey different perspectives. Only develop those you think are essential.
- Content is more important than representation.
- Know your models and adapt locally.



Summary