AGILE FUNDAMENTALS

ThoughtWorks®

SHORT INTRODUCTIONS

- Who are you?
- What do you do?
- What is your main learning objective?

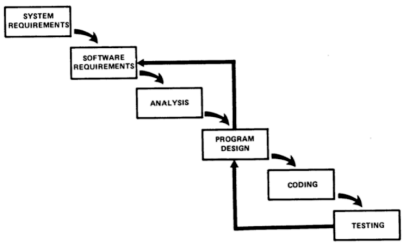
GROUND RULES

- Break times
- Restroom
- Start on time
- Phones off or on silent
- Laptops down
- No side conversations / one conversation at a time
- Criticize ideas, not people
- No hierarchy
- Respect

QUESTIONS?

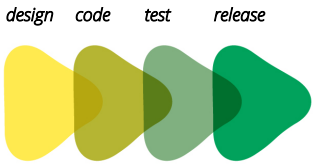
ThoughtWorks®

THE "WATERFALL" DIAGRAM

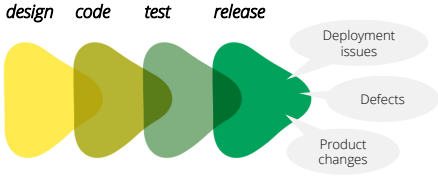


[1] Royce, Winston. "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON26 (August): 1–9. 1970.
[2] Bell, Thomas E., and T. A. Thayer. "Software requirements: Are they really a problem?", Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, 1976.

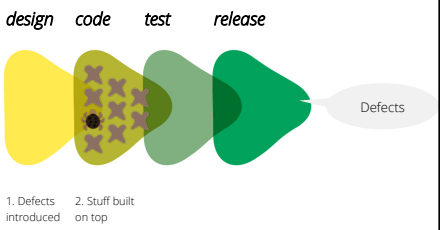
A WATERFALL RELEASE



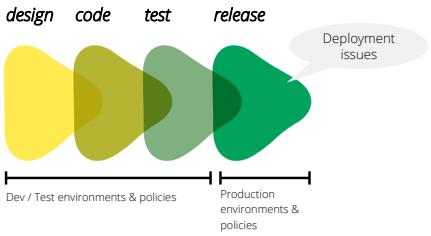
A WATERFALL RELEASE



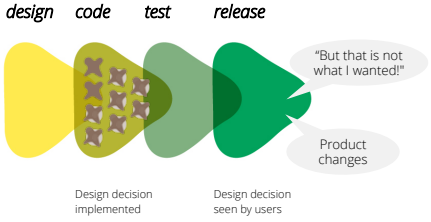
A WATERFALL RELEASE



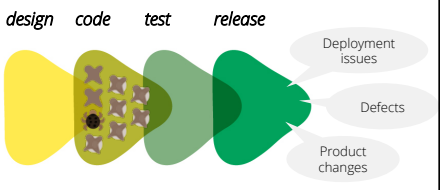
A WATERFALL RELEASE



A WATERFALL RELEASE



A WATERFALL RELEASE



ThoughtWorks®

AGILE SOFTWARE DEVELOPMENT

WHAT IS AGILE?

*A way of building software that is all about planning for, and assuming **change***

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

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.

TWELVE PRINCIPLES OF AGILE SOFTWARE

1. Priority 1: Satisfy customer via early, continuous, valuable delivery
2. Welcome changing requirements, even late in development
3. Deliver working software frequently
4. Business people and developers must work together
5. Build projects around motivated individuals.
6. Face-to-face conversation is most effective way to communicate.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
9. Continuous attention to technical excellence and good design
10. Simplicity (art of maximizing the amount of work not done)
11. Emergent design and requirements from self-organizing teams
12. Regular reflection on how to become more effective

IT IS NOT EITHER-OR

- Scrum
- XP (eXtreme Programming)
- Lean-kanban
- Hybrid (e.g., "Scrumban")

ThoughtWorks®

WHAT AGILE IS *NOT*

AGILE IS NOT...

- A "silver bullet"
- One size fits all
- Go through the motions and mechanics, and then "we are agile"
- Ad-hoc with no plan

ThoughtWorks®

WHAT AGILE IS

- **Incremental**
  Build software gradually
  Show progress

da Vinci as an incremental painter

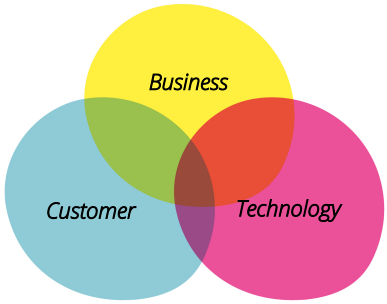
- **Incremental**
  Build software gradually
  Show progress
- **Iterative**
  Multiple iterations and releases

da Vinci as an iterative painter

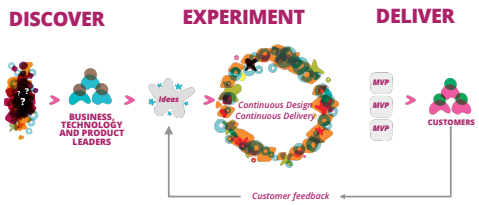
- **Incremental**
  Build software gradually
  Show progress
- **Iterative**
  Multiple iterations and releases
- **Adaptive**
  Incorporate feedback and changes to goals
  Lessons learned

da Vinci as an incremental
iterative
adaptive painter

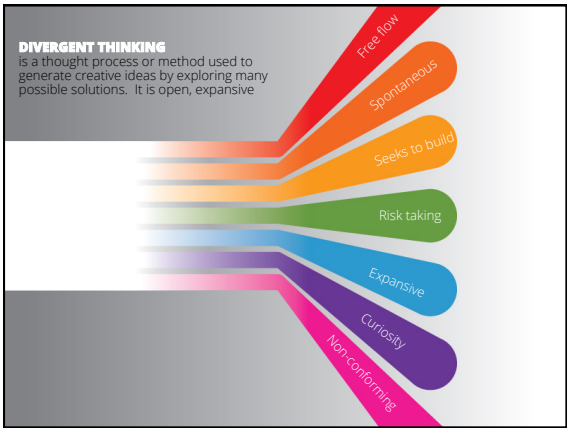
- **Incremental**
  Build software gradually
  Show progress
- **Iterative**
  Multiple iterations and releases
- **Adaptive**
  Incorporate feedback and changes to goals
  Lessons learned
- **Collaborative**



OUR APPROACH

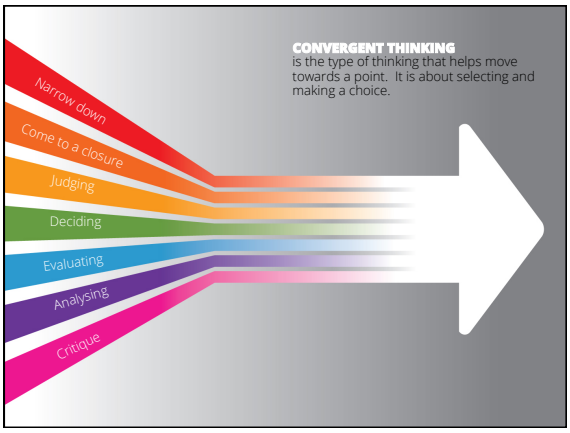


DIVERGENT THINKING
is a thought process or method used to generate creative ideas by exploring many possible solutions. It is open, expansive

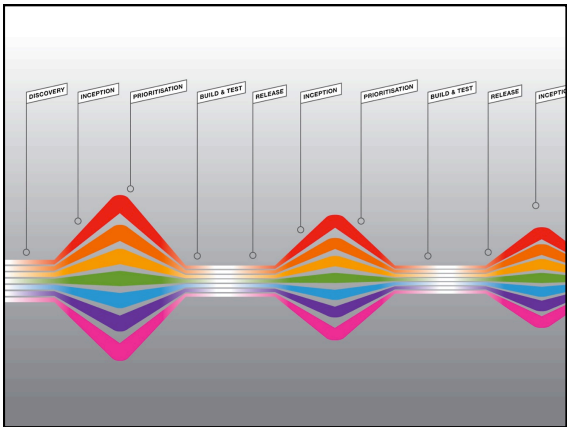


SUCCESSFULLY DIVERGENT

- Create the right environment
- Defer judgment… evaluate later
- Challenge someone to modify their idea, to build on it and improve it
- Ask "what else?"
- Make connections with others ideas, spring board off them, build on them

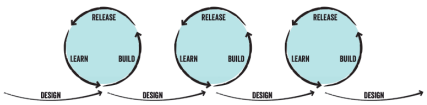

CONVERGENT THINKING
is the type of thinking that helps move towards a point. It is about selecting and making a choice.
Narrow down
Come to a closure
Judging
Deciding
Evaluating
Analysing
Critique


SUCCESSFULLY CONVERGENT
Don't stop with the first good idea
Eliminate bad ideas quickly
Close everything you open
Have agreed criteria for your selection
The problem is well understood

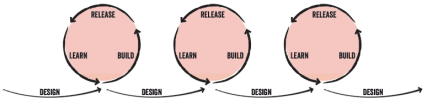

DISCOVERY
INCEPTION
PRIORITISATION
BUILD & TEST
RELEASE
INCEPTION
PRIORITISATION
BUILD & TEST
RELEASE

DESIGN THINKING AND AGILE

Design Thinking *is the process of continually designing and learning from people (consumers, customers, patients, staff)*

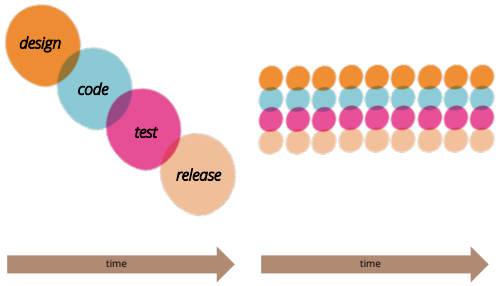


Agile Software Development *is a process of rapid software design and build cycles which means constant releases to customers.*



DESIGN THINKING AND AGILE

WATERFALL VS. AGILE APPROACH

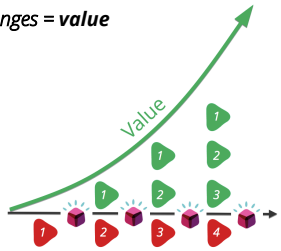


WATERFALL VS. AGILE APPROACH

*unreleased changes = **risk***

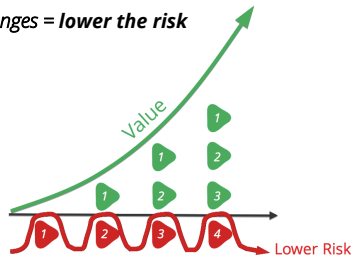
WATERFALL VS. AGILE APPROACH

*released changes = **value***

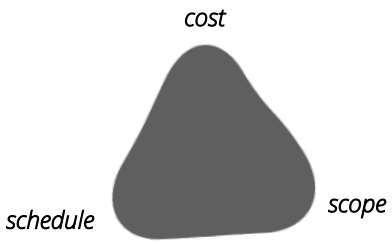


WATERFALL VS. AGILE APPROACH

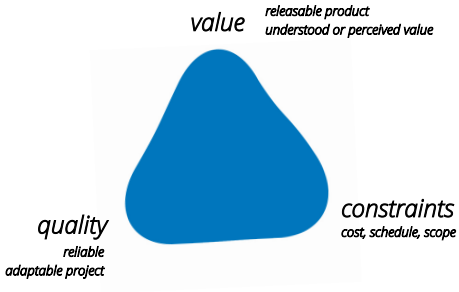
*released changes = **lower the risk***



THE TRADITIONAL TRIANGLE



THE AGILE TRIANGLE

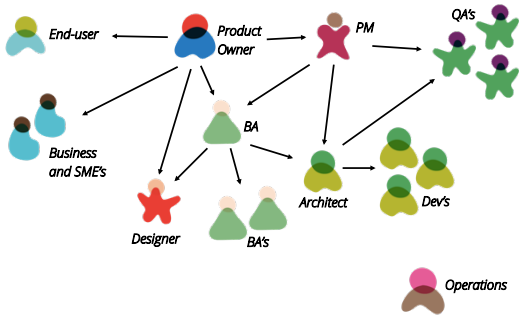


Both management and technical practices are important.

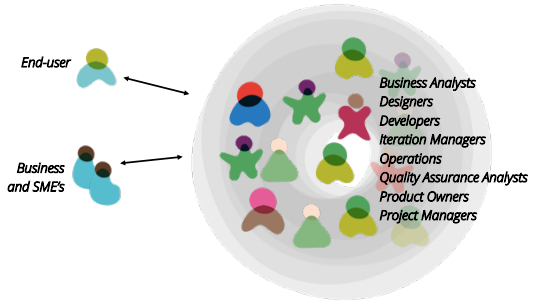
To be truly agile, the software itself needs to be agile.

- Representation across roles
- No specialized teams
- Typically co-located

TRADITIONAL SILOS



EMPOWERED TEAMS



QUESTIONS?

ThoughtWorks®

USER STORIES

- A tool for **iterative** development
- Represents a **unit** of work that should be developed
- Help **track** that piece of functionality's lifecycle
- Placeholder for a **conversation**

WHY SHOULD I USE THEM?

- It is a piece of customer-visible functionality written in common language
  Universally understood
- Ensure you only build things for a reason
  Prevents waste
- Simple and flexible
  Minimum overhead
- Proven way of gathering requirements on agile projects
  Effective teams

ThoughtWorks®

As a **<role>**

I want to **<business goal>**

So that **<value>**

INVEST IN STORIES

I	Independent	*No overlap – order is ok!*
N	Negotiable	*No contract. Details can change.*
V	Valuable	*Incremental benefit to something.*
E	Estimable	*Relative size to other stories.*
S	Small	*Shouldn't be bigger than an iteration.*
T	Testable	*Should be able to tell when it is done.*

INVEST IN STORIES

V Valuable

Someone is benefiting from what we are building

Why are we building it?

INVEST IN STORIES

T Testable

Now we know what to do and know when we will be done

INVEST IN STORIES

S Small

Just enough to get feedback and avoid waste

INVEST IN STORIES

I Independent

Should be able to realize the value of each single story.

Distance from the airport can be in another story.

INVEST IN STORIES

N Negotiable

INVEST IN STORIES

E Estimable

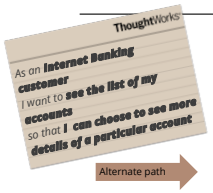
WHAT IS IN A USER STORY

- "As a … I want … So that …"
- Acceptance Criteria
- Prototype
- User Interface design
- Other text/images/content to provide context

ACCEPTANCE CRITERIA

- Tell you the story is finished Acceptance tests tell you the system is working
- A story can have one or multiple AC's
- It informs the criteria to the tests that will be implemented or executed
- Common format:
  Give <context>
  When <event>
  Then <outcome>

ACCEPTANCE CRITERIA



Given the customer has one transaction account and one credit account
When they have completed logging in
Then the screen should show the names and numbers of the two accounts sorted in account number order

Given the customer has just one transaction account
When they have completed logging in
Then the screen should show the name and number of the account

Alternate path

Given the customer has no accounts
When they have completed logging in
Then the screen should show a message stating that no accounts are available

Alternate path

Given the customer has more than 20 accounts
When they have completed logging in
Then the screen should show the first 20 accounts (in account number order) only

Bad path

Given the customer has some accounts
And they have completed logging in
When the system cannot retrieve the account details
Then the screen should show an error message with associated code and details to contact for support

WHAT IS *NOT* IN A USER STORY

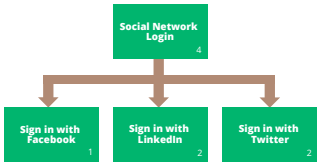
- Unit and Integration tests
- Functional/Acceptance tests
- Documentation
- Specifications

STORY DETAILS AT THE RIGHT TIME

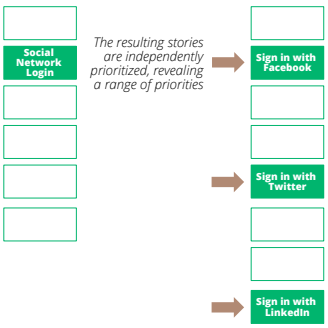
STORY BREAKDOWN

- Identify opportunities to split individual stories into one or more smaller independent stories, in order to:
  - Make stories an appropriate size for delivery
    eg. small enough that they are easily understandable, estimable and will give a good indication of progress
  - Identify differences in priority
    to ensure only the highest priority work is completed first

SPLITTING BY FEATURE

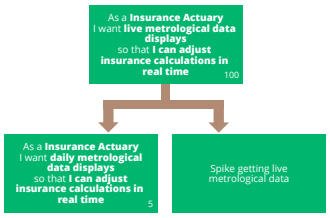


SPLITTING BY PRIORITY

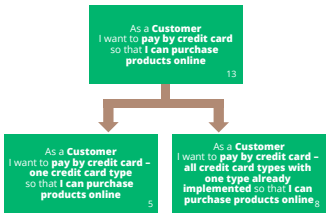


SPLITTING BY VALUE

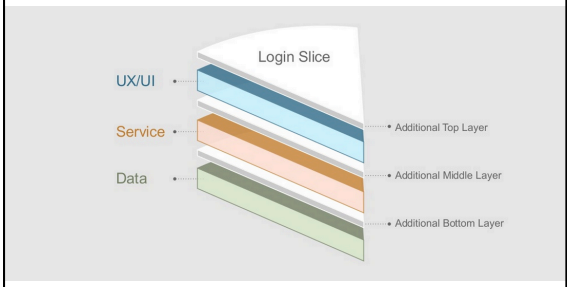
SPLITTING BY RISK



SPLITTING DEPENDENCIES



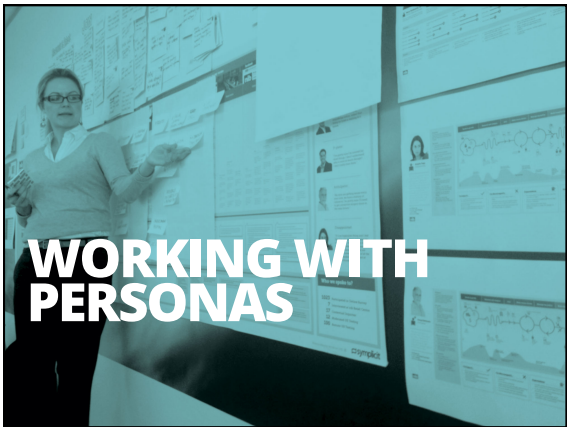
SLICING VERTICALLY



Made by Ben Clay

SLICING VERTICALLY

Made by Ben Clay



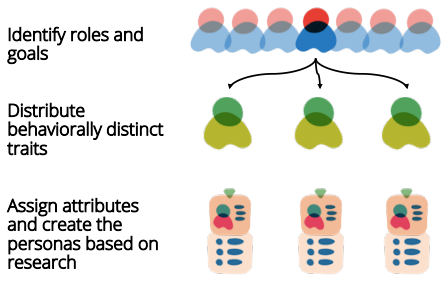
PERSONAS

- Concrete characterization of a user group
- Archetype / representation / profile
- Description of end users based on research
- Reference document
- It can be used a synthesis of different profiles

HOW TO DEVELOP PERSONAS

1. Conduct research
2. Identify and cluster the behavioral variables
3. Identify trends
4. Create a persona for each of the trends
5. Create scenarios and tell stories
6. Create personas documentation

HOW TO DEVELOP PERSONAS



<Persona name>

"Insert quote that characterises this persona in one sentence."

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Name <name>
Type <type>
Role <role>

Motivations
- Excepteur sint occaecat cupidatat non.
- Sunt in culpa qui officia deserunt mollit anim id est laborum.
- Excepteur sint occaecat cupidatat non.

Goals
- Excepteur sint occaecat cupidatat non.
- Sunt in culpa qui officia deserunt mollit anim id est laborum.
- Excepteur sint occaecat cupidatat non.

Pain points
- Excepteur sint occaecat cupidatat non.
- Sunt in culpa qui officia deserunt mollit anim id est laborum.
- Excepteur sint occaecat cupidatat non.

Common profiles
- Excepteur sint occaecat cupidatat non.
- Sunt in culpa qui officia deserunt mollit anim id est laborum.
- Excepteur sint occaecat cupidatat non.

Behaviours

ROLES VS. PERSONAS

- Personas can easily become a reflection of roles within a system, usually a result of not digging deep enough to identify the different behavior patterns within each role
- Personas should be able to identify different behaviors of roles
- Danger of role-based personas is that there is often a temptation to create them based existing team knowledge of the system

QUESTIONS?

ThoughtWorks®

WHY ESTIMATE?

- Provides a measure of progress
- Helps with prioritization
- Estimation discussions are very helpful for defining an approach to delivering an individual story
- BUT:
  - Time consuming for large numbers of stories
  - Not a measure of time

WHAT TO ESTIMATE

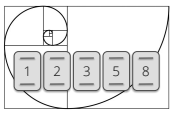
- Don't try and estimate all the stories in a project
- Have a prioritized backlog and estimate highest priority first
- Ensure there are estimated stories for the next Iteration
- Stories that are likely to be difficult

WHAT TO ESTIMATE

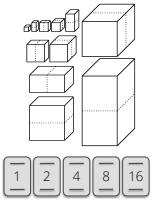
- A common view of what needs to be complete for a story to be counted as done and to include its story points in velocity
- Commonly includes: functional code completed, tests passed, acceptance test passed, and deployed to production environment
- Definition should be discussed and agreed as part of inception
- Should define a situation where the team will not have to work on the story again

HOW TO ESTIMATE

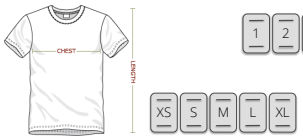
Fibonacci



Power of Two



T-Shirt size

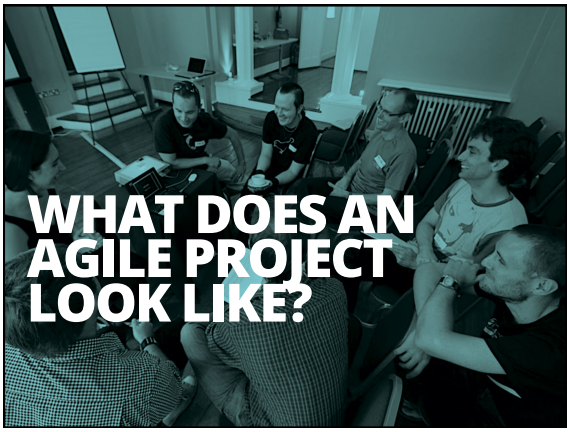


GOOD PRACTICES ON ESTIMATION

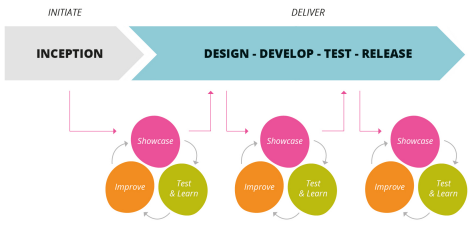
- A story should be deliverable from start to finish with in iteration. If not split the story
- The smallest story should represent a single story point
- Those involved in delivery should be part of estimation
- Participants should discuss the story until they all agree on an estimate
- There should be sufficient understanding of how the story will be delivered
- Have spikes when technical uncertainty is too high
  - Solve hard technical problems through time-boxed and real experiments without tampering with production code

QUESTIONS?

ThoughtWorks®



WHAT DOES AN AGILE PROJECT LOOK LIKE



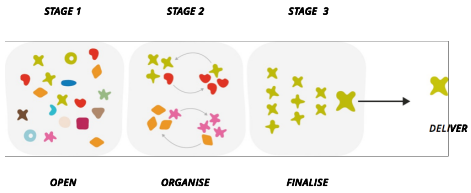
HOW INCEPTIONS FEEL

- Highly collaborative and inclusive
- Time-boxed and rapid, focused on doing 'just enough'
- Feedback-driven and highly adaptive
- Is not upfront analysis
- Highly visual and workshop oriented to help evolve a vision for the project.

INCEPTION OBJECTIVES

Build a shared understanding of project vision and goals	Evaluate high-level scope and core processes
Evaluate key risks, issues and constraints in delivering to the end goal	Establish technical and testing approach

FAST AND COLLABORATIVE

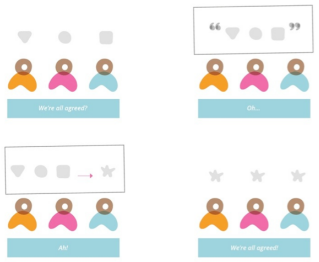


OUR JOURNEY

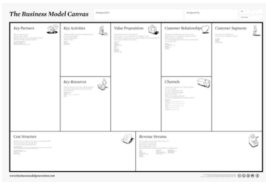
OUTCOMES

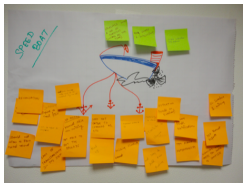
- All artifacts necessary to get started
- A prioritized list of features that deliver the most business value
- Consensus about the future state
- A shared vision of the solution
- Jointly produced system requirements
- A realistic, achievable plan for an early release of business value.

CREATE A SHARED VISION



UNDERSTAND BUSINESS VISION



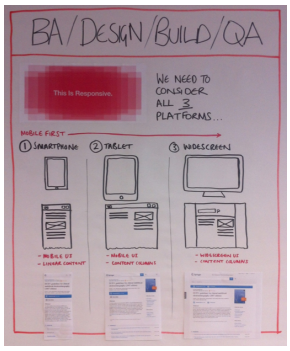


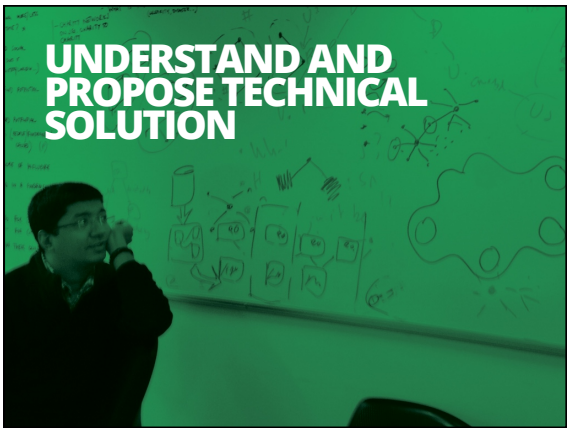
(Note to reviewers - a new, crafted business model canvas will replace this one.)

USER EXPERIENCE DESIGN

- Discovery
  - User Personas
- Concept
  - User Journey
  - Information Architecture
  - Collaborative Design
  - Wireframes
- Visual Design

COLLABORATIVE DESIGN APPROACH

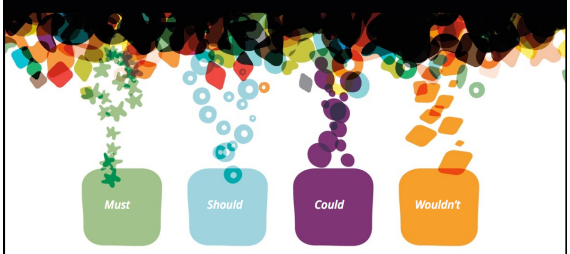




TECHNICAL APPROACH

- Technical visioning
- Architecture
- Proposed testing approach
- Addressing Non Functional Requirements (NFRs)

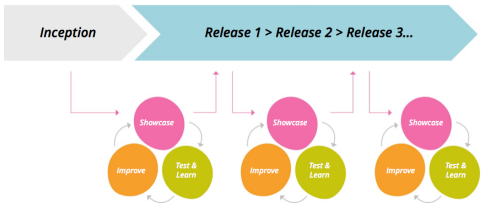
PRIORITISE



We capture…

RISKS
ASSUMPTIONS
ISSUES
DEPENDANCIES

PREPARE RELEASE PLAN



QUICK RECAP

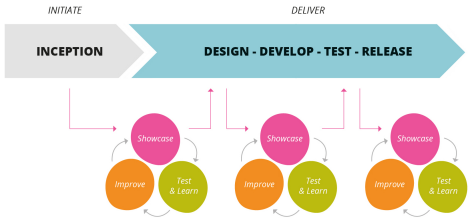
INCEPTION DELIVERABLES

- Business Vision and Project Objectives
- Prioritised High Level Requirements (Stories & NFRs)
- Technical Approach
- Visual Design Mock-ups
- Release Plan
- RAIDs & Communication Plan
- Next Steps

QUESTIONS?

ThoughtWorks®

WHAT DOES AN AGILE PROJECT LOOK LIKE



ThoughtWorks®

As a **customer**
I want to **be able to sign in using my Facebook account**
so that **I don't need to remember *another* username and password**

INITIAL REQUIREMENTS

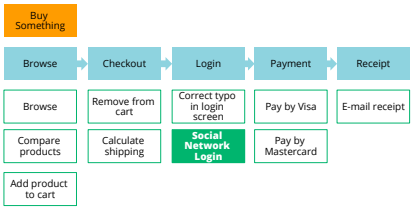
- *project inception*
- *unfinished from previous projects*
- *ad-hoc conversations*
- *analysis activities*
- *non-functional reqs*

- Initial project requirements come from a wide variety of sources. Perhaps the most common source is inception workshops that involve the core delivery team and stakeholders
- Requirements will often vary greatly in level of details ... from loosely-defined **epics** through to **stories**, down to very well-defined **bugs**

PRODUCT BACKLOG

- These initial requirements form our Product Backlog
- At the simplest, just a prioritized/ranked list of requirements
- The Product Backlog is owned and maintained by the Product Owners with assistance of Analysts over the lifespan of the project
- Product Backlogs evolve over the course of the project (they are not fixed after being defined up front!)

STORY MAP



NON-FUNCTIONAL REQUIREMENTS

some examples:

Auditability	Authentication	Authorization
Availability	Help	Localization
Performance	Portability	Regulatory
Scalability	Security	Usability

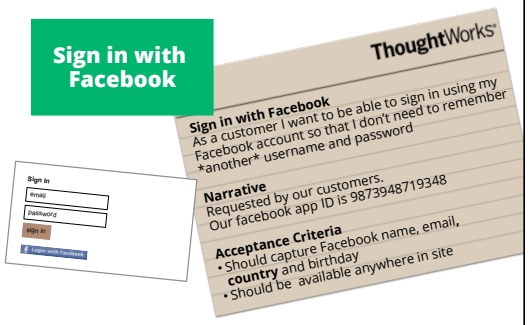
NFR: WHAT TO DO WITH THEM?

- Discover which ones matter
- New stories (a mind map may help)
- Acceptance criteria
  - in some stories
  - in all stories
- Constraints
  - "Constraint cards"
- Review the list of 'ilities' in a facilitated session

ANALYSIS AND UX ELABORATION

- Analysts, UXers and Product Owners progressively elaborate Stories for development – high priority first!
- Stories are picked up for elaboration far enough ahead of when they are expected be implemented to ensure detail is ready, but no so far in advance that it's hard to change course
- Elaboration activities include:
  - Story breakdown
  - Finishing User Stories, Acceptance Criteria and Narratives
  - Prioritization
  - Estimation

ANALYSIS AND UX ELABORATION



ESTIMATION AND PRIORITIZATION

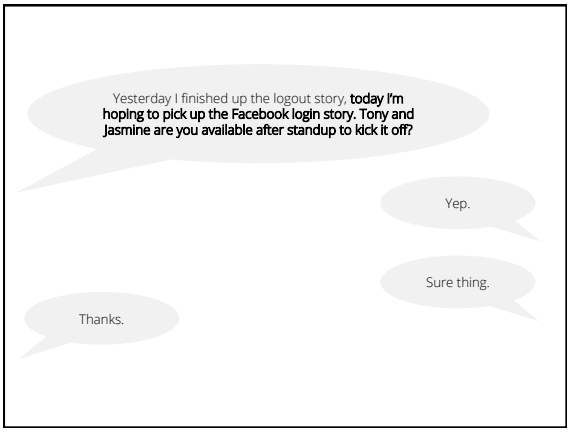
- Periodically (every couple of iterations) prioritization should be revisited with Product Owners to see if any priorities have shifted
- Similarly if elaboration identifies substantial changes in scope or potential effort, estimates should be revisited with the Core Team
- This activity of revisiting the backlog for estimation, prioritization and add/remove stories based on recent learning is sometimes known as "backlog grooming"

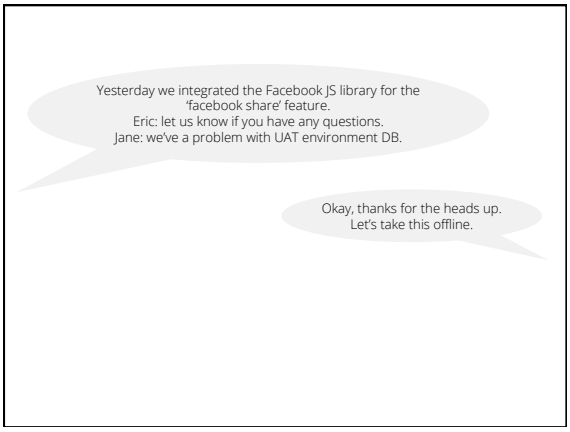
ITERATION PLANNING MEETING

- Agenda
  - Confirm priorities
  - Calculate expected team capacity
  - Overview of high priority stories
  - Development team indicates which stories they can commit to that iteration

STAND-UP MEETING

- Team meets for 15 minutes daily to provide a status update among team members:
  - What did I accomplish yesterday?
  - What will I do today?
  - What obstacles are impeding my progress?
- Goal is to highlight potential challenges and identify ownership to coordinate efforts to resolve difficult or time-consuming issues
- It is not a goal to resolve those issues in this 15 minute meeting





STORY KICK-OFF

- Product Owner or Analyst introduces the rest of the team to the story and answers any questions that come up at this point (may tweak story details)
- Developers describe how they intend to approach the solution
- Testers describe how they intend to approach the testing
- Designers talk about how the UX should be like
- Meeting ends when everybody is comfortable (5-30min)

DEVELOPMENT

- Developers work alongside UX, Analyst, Product Owners and Testers as required. Here conversations fill in any remaining details and any options (such as different solutions) are evaluated
- Testers may also prepare test cases and annotate on the story
  - Test cases are not acceptance criteria!
  - Test cases will include things not covered in acceptance criteria
- Development continues until developers are confident the acceptance criteria are met, and an environment can be provided for testing
- Do regular desk checks with the rest of the team

HANDOVER TO TEST

- Developer hands over story to Tester
  - If available, analyst, designers and product owners might be included
- Demonstrates functionality against the acceptance criteria in the test environment (if cant be demonstrated… it isn't ready!) (5-15 min)

TESTING

- Tester finalizes and executes test cases for the story. Will seek additional clarification from Product Owners, Analysts, Designers or Developers
- Tester will do further exploratory testing to test the edge cases of the solution
- Any issues identified that would prevent the story from meeting acceptance criteria should be properly handled

STORY ACCEPTANCE

- Analyst or Tester demonstrates the solution to Product Owners to answer two questions:
- Does this solution meet the acceptance criteria (as agreed)… are we done on this story?
- Based on now seeing the working software, does this inspire other changes to this feature?
  All new ideas → new user stories (unless the delivery team agrees) Product Owners prioritize new stories in existing backlog
- Demonstration often involves walkthrough of Given…When… Then acceptance criteria, as well as some exploratory use by Product Owners
- It should not be the first time the Product Owners see the story implemented

SHOWCASE

- Product Owners and Team demonstrate the stories that have been accepted as 'done' during the iteration to the full set of users and stakeholders
  ie. everyone with any stake in the project
- Feedback is gathered
  All changes, ideas → new user stories (product owners prioritize)

RELEASE

- The earlier the product under development reaches the hands of the end-users the better – it provides an opportunity to measure and respond to real usage of the system while there is still time to adjust

RETROSPECTIVE

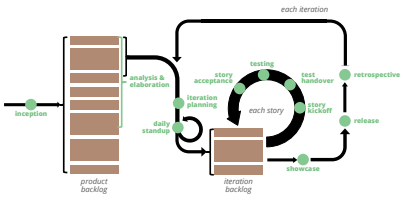
- Team brainstorms on sticky-notes:
  - What went **well** with the delivery process?
  - What went **not so well** with the delivery process?
  - What **ideas** do we have about things we can do differently?
  - What still **puzzles** us about the delivery process?
- Similar thoughts are combined into groups (sticky-notes placed together)
- Team votes on the groups to identify which they want to discuss most
- Team discusses top-voted groups, identifies and formulates actions: **who** will do **what** by **when**

RETROSPECTIVE PRIME DIRECTIVE

Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

(Norm Kerth)

WRAPPING UP THE ITERATION



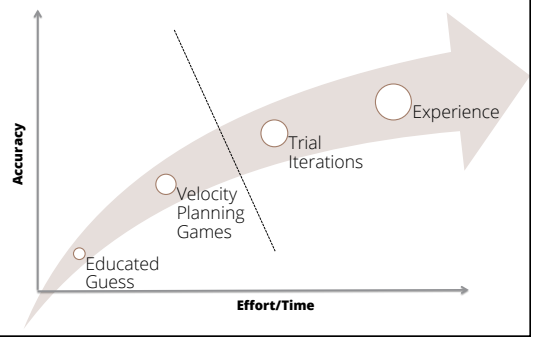
QUESTIONS?

ThoughtWorks®

ThoughtWorks®

PLANNING AND TRACKING

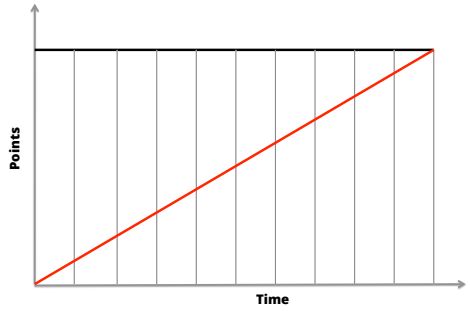
HOW TO DERIVE A PLANNING VELOCITY



HOW LONG WILL THE PROJECT TAKE?

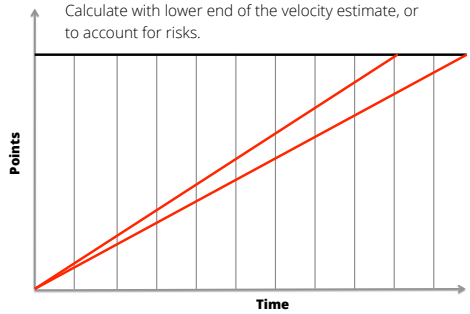
1. Run Velocity Planning games
2. Find the total number of story points
3. Find raw velocity per iteration
   estimated velocity per pair X steady state pairs
4. Find number of iterations
   total number of story points / estimated velocity

RAW VELOCITY



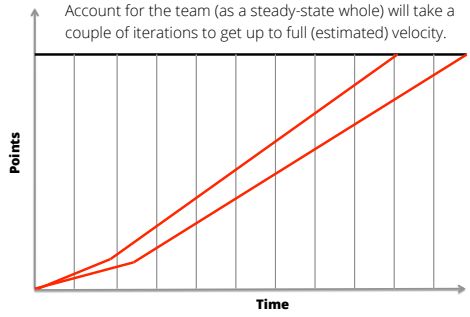
VELOCITY ADJUSTMENT: RISK

Calculate with lower end of the velocity estimate, or to account for risks.



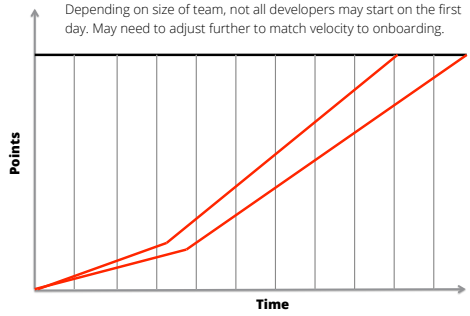
VELOCITY ADJUSTMENT: RAMP UP

Account for the team (as a steady-state whole) will take a couple of iterations to get up to full (estimated) velocity.

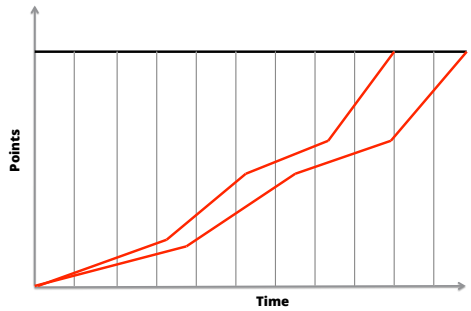


VELOCITY ADJUSTMENT: ONBOARDING

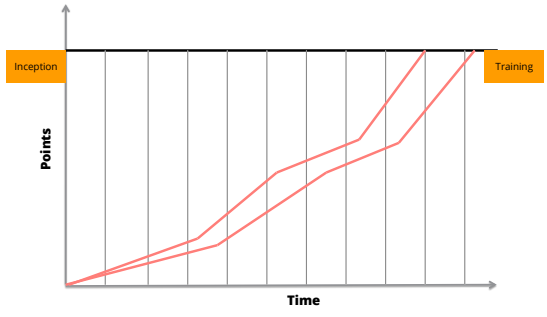
Depending on size of team, not all developers may start on the first day. May need to adjust further to match velocity to onboarding.

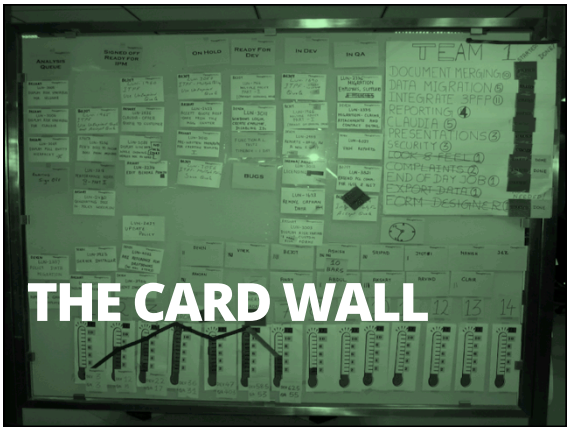


VELOCITY ADJUSTMENT: TIME-OFF



NON-DEVELOPMENT TIME





THE CARD WALL

QUESTIONS?

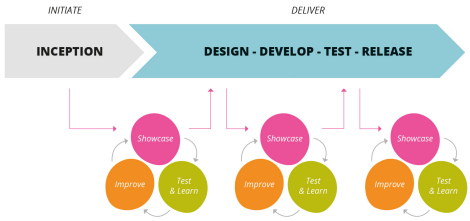
ThoughtWorks®

ThoughtWorks®

WRAPPING UP ON THE PROJECT LIFE

IT STARTS WITH THE INCEPTION...

...AND ITERATES THROUGHOUT THE PROJECT



QUESTIONS?

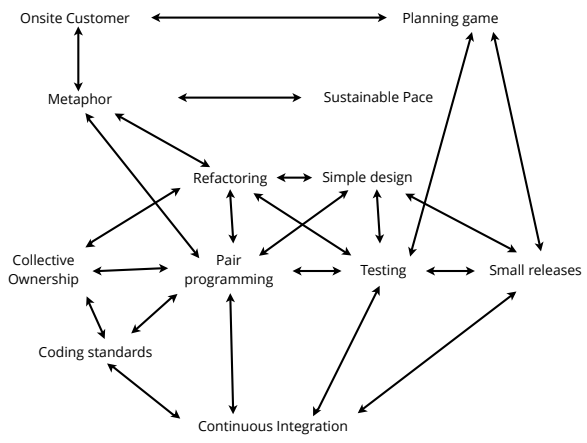
ThoughtWorks®

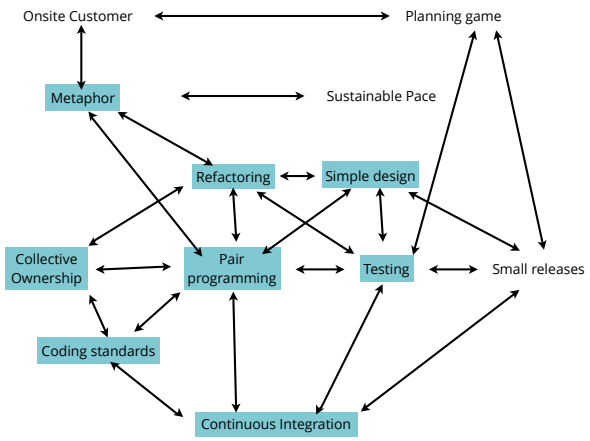
ThoughtWorks®

DEVELOPMENT PRACTICES

AGENDA

- Agile software development
- Test Driven Development
- Refactoring
- Evolutionary Architecture
- Continuous Integration
- Collective Ownership





4

TEST DRIVEN DEVELOPMENT

5

Write a failing test *Make it pass* *Make it better*

Red *Green* *Refactor*

6

TDD helps communicate intentions

TDD focuses on behaviour, not implementation

REFACTORING

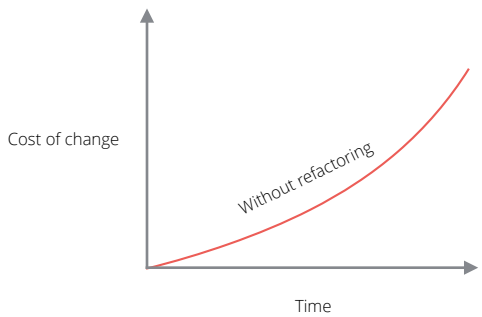
Refactoring is a behaviour preserving transformation

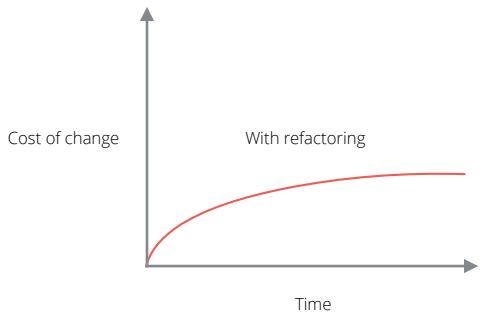
Functionally, the software is the same

Refactoring improves maintainability & extensibility

Refactoring vs redesign

WHY REFACTOR?





16

WHEN TO REFACTOR

- Adding functionality
- Fixing a bug
- Doing a code review

17

WHEN NOT TO REFACTOR

- Code is too messy
- Near a deadline

18

DEMO

WHY PAIR

- Fewer defects
- Less rework
- More creativity
- Easier to maintain

PAIRING STYLES

- Ping-pong
- Driver Navigator

PAIRING MYTHS

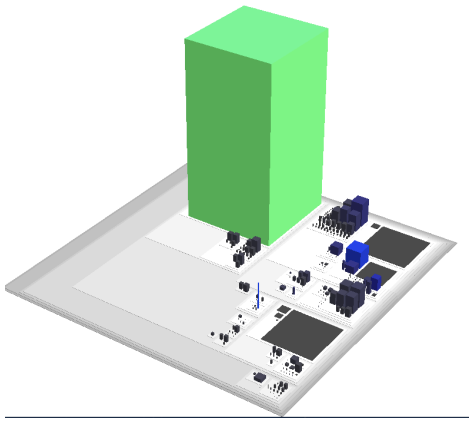
- It's distracting!
- It's costly!
- It hurts morale!
- Will it slow me down?

CODE SMELLS

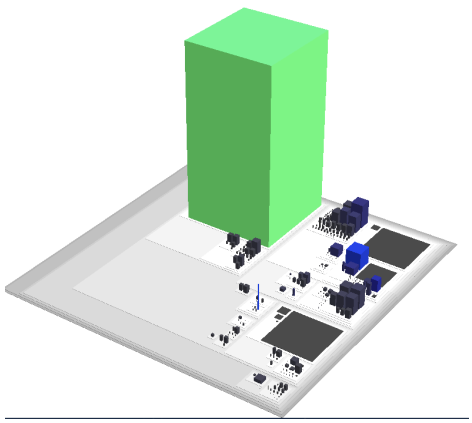
CODE SMELLS

- Long methods
- Lots of parameters
- Code comments
- Magic numbers
- Duplication

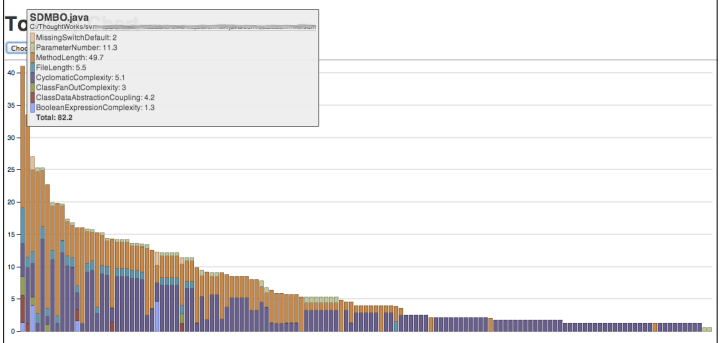
BIG SCARY CLASS



ANAEMIC MODEL



TOXIC CODE



Hover over the bars to see detailed information about the score for the class.

The y-axis is fixed with 41 as a maximum to make it easier to compare charts for different code-bases.

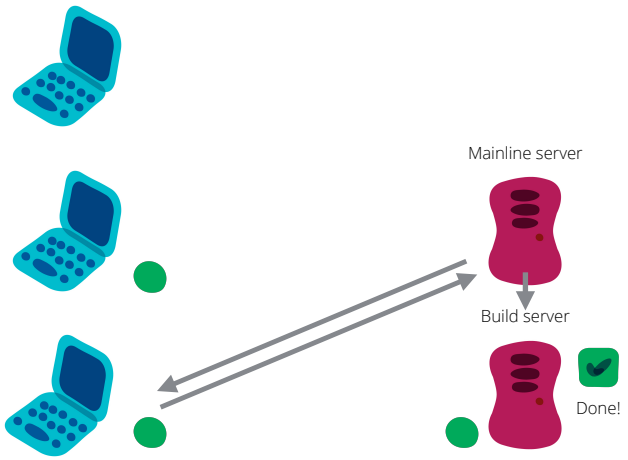
EVOLUTIONARY ARCHITECTURE

Growing software, guided by tests

EVOLUTIONARY ARCHITECTURE

- Incremental, not Big Design Up Front
- Simple design
- YAGNI

CONTINUOUS INTEGRATION

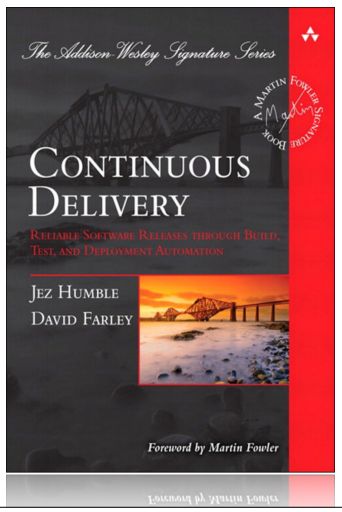


WHY PRACTISE CI?

- Quick feedback on problems
- Gets the most out of automated testing
- Tests, builds deployments using production-like setup
- Reduces waste due to manual integration
- Provides a safety net allowing changes to be made with confidence

CONTINUOUS INTEGRATION PRACTICES

- Keep the build fast: < 10 minutes
- Keep the build green
- Fix breakages within 10 minutes
- Commit as frequently as possible
- Avoid branching during development
- Trunk based development with feature toggles



CONTINUOUS DELIVERY

- Automate almost everything
- Everything in version control
- Repeatable process for releasing software
- Build quality in
- “Done” means released
- Everyone is responsible for delivery

37

38

AGILE DEVELOPERS

- Can work with any part of the stack
- Can work on any story
- Usually have deep expertise in certain areas
- “T-Shaped”

39

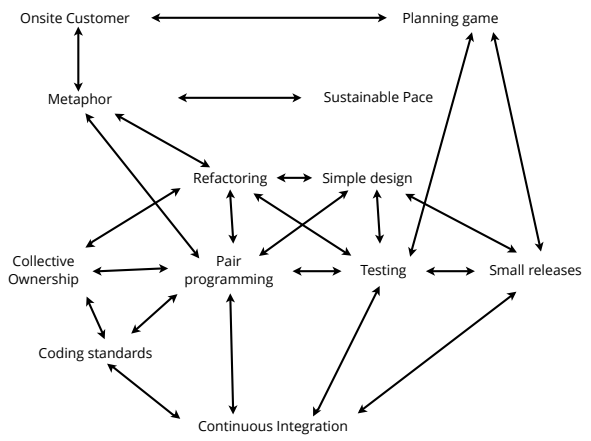
SUMMARY

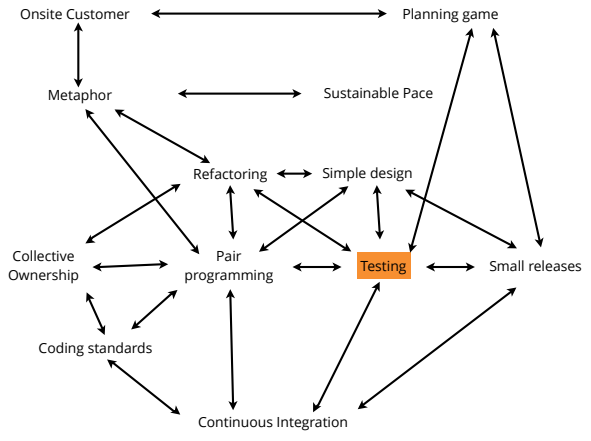
- Test Driven Development
- Automated Testing
- Refactoring
- Evolutionary Architecture
- Continuous Integration
- Continuous Delivery

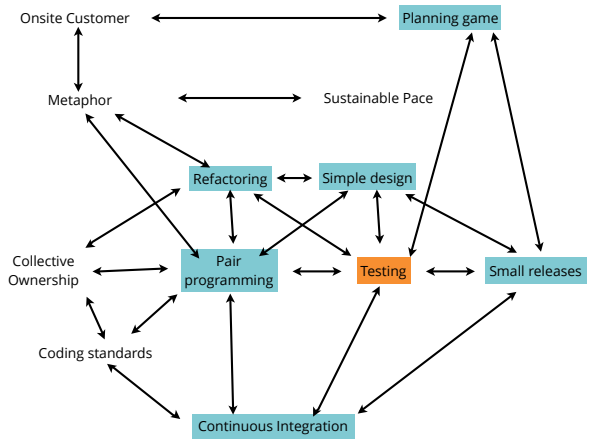
QUESTIONS?

ThoughtWorks®

QUALITY



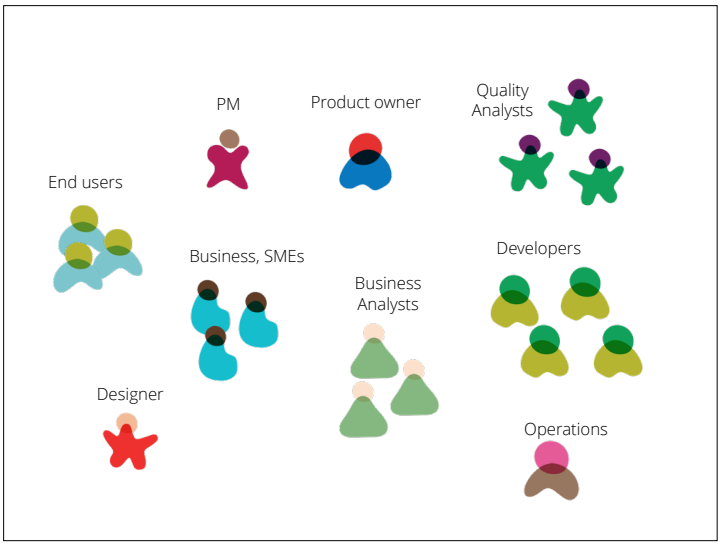




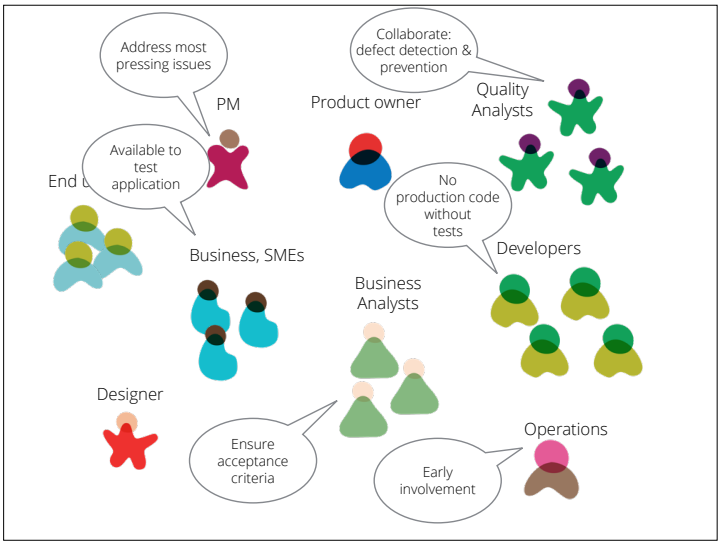
WHAT IS QUALITY?

- Provide confidence in a product's suitability
- Determined by intended users
- Product quality *vs* Quality product

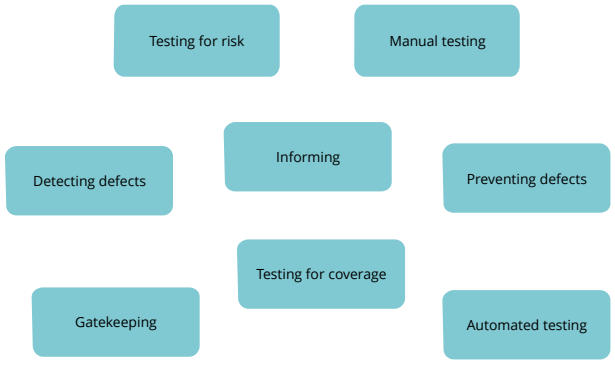
WHO OWNS QUALITY?



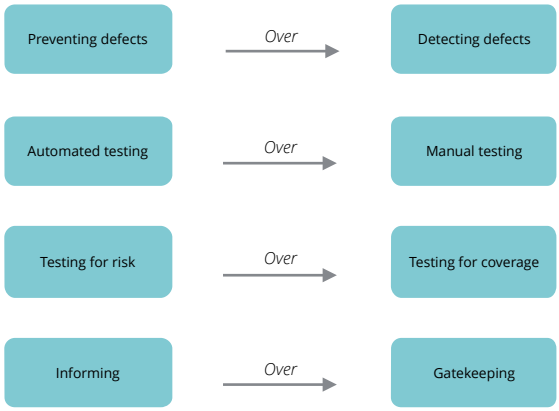
HOW WE FOCUS ON QUALITY



THE AGILE QA MANIFESTO



THE AGILE QA MANIFESTO



INFORMING OVER GATEKEEPING

QUALITY IS A PART OF EVERYTHING

- Business analysts write acceptance tests
- Developers write unit and integration tests
- Testers automate acceptance tests
- Regression suite run as part of every build

QA HIGHLIGHTS RISK

DEFECT PREVENTION

CREATING ACCEPTANCE CRITERIA

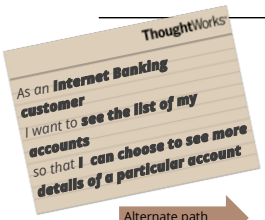
S	Specific	Explicitly defined, definite
M	Measurable	Can observe, quantify
A	Achievable	Can be done
R	Relevant	Connected to the story
T	Timely	When is outcome observed?

16

WRITING ACCEPTANCE CRITERIA

17

ACCEPTANCE CRITERIA



Given the customer has one transaction account and one credit account
When they have completed logging in
Then the screen should show the names and numbers of the two accounts sorted in account number order

Given the customer has just one transaction account
When they have completed logging in
Then the screen should show the name and number of the account

Alternate path

Given the customer has no accounts
When they have completed logging in
Then the screen should show a message stating that no accounts are available

Alternate path

Given the customer has more than 20 accounts
When they have completed logging in
Then the screen should show the first 20 accounts (in account number order) only

Bad path

Given the customer has some accounts
And they have completed logging in
When the system cannot retrieve the account details
Then the screen should show an error message with associated code and details to contact for support

64

AM I DONE?

THE WHOLE TEAM APPROACH

POTENTIAL ISSUES

Lack of face to face communication?

22

Colocation

23

No shared responsibility towards quality?

24

Testing everywhere: acceptance criteria, developer testing, automated testing

Misaligned metrics

Measure throughput

Timing of testing / schedule pressure

Testing in parallel with development

THE TESTERS BILL OF RIGHTS

- Ask questions of customers and developers
- Bring up quality process issues any time
- Ask for, and receive help from anyone
- The tools needed to do the job

TIPS FOR COLLABORATION

- Not us vs. them
- No excuses
- Testing is not just for testers!

TESTING IS NOT JUST FOR TESTERS

- Automation support
- Environment setup
- Management support
- End-to-end scenario creation

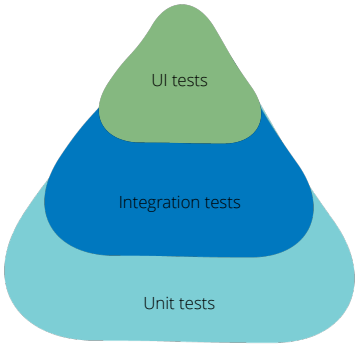
EXERCISE: 99 BALLOONS

AUTOMATED TESTING AND THE CENTRAL ROLE OF QUALITY

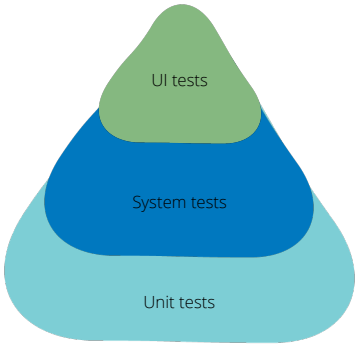
AUTOMATION

HOW MUCH SHOULD YOU AUTOMATE?

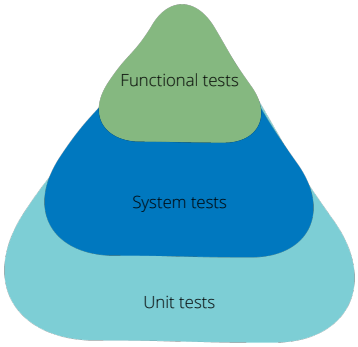
THE TESTING PYRAMID



THE TESTING PYRAMID



THE TESTING PYRAMID



UNIT TESTING / TDD

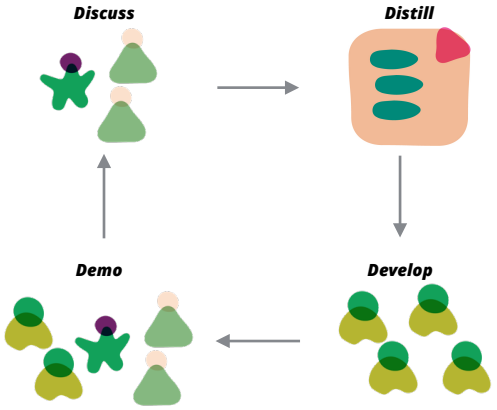
FUNCTIONAL TESTING

USER ACCEPTANCE TESTING

REGRESSION TESTING

SYSTEM INTEGRATION TESTING

ACCEPTANCE TEST DRIVEN DEVELOPMENT



AGILE AUTOMATION

WHY IS AUTOMATION IMPORTANT?

- Fast feedback
- Confidence to the development team
- Almost impossible to manually test a rapidly changing system

Release 1 *Release 2* *Release 3*

AUTOMATION IN AN AGILE TEAM

- Automation in parallel with development
- Automated acceptance tests
- Automated tests add to a regression suite
- Test executed as part of continuous integration

WHEN DOES AUTOMATION FAIL?

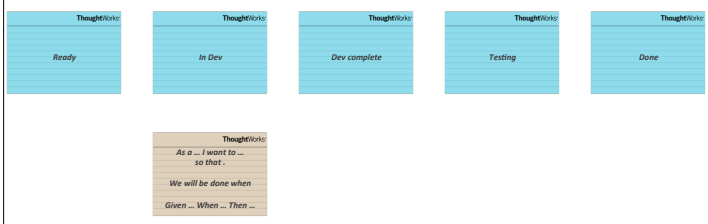
- Overengineering test scripts
- Testing everything through functional tests
- Wrong choice of tools

WHAT IS WRONG WITH TRADITIONAL TOOLS?

- Test-last workflow does not fit in with the agile way
- Tools tend to encourage record & play
- Needs test automation specialist
- Tools do not encourage collaboration

DEFECTS

WHEN TO RAISE A DEFECT



ThoughtWorks
Ready

ThoughtWorks
In Dev

ThoughtWorks
Dev complete

ThoughtWorks
Testing

ThoughtWorks
Done

ThoughtWorks
As a ... I want to ...
so that .
We will be done when
Given ... When ... Then ...

ThoughtWorks
Ready

ThoughtWorks
In Dev

ThoughtWorks
Dev complete

ThoughtWorks
Testing

ThoughtWorks
Done

ThoughtWorks
As a ... I want to ...
so that .
We will be done when
Given ... When ... Then ...

ThoughtWorks
Ready

ThoughtWorks
In Dev

ThoughtWorks
Dev complete

ThoughtWorks
Testing

ThoughtWorks
Done

ThoughtWorks
As a ... I want to ...
so that .
We will be done when
Given ... When ... Then ...

ThoughtWorks
Ready

ThoughtWorks
In Dev

ThoughtWorks
Dev complete

ThoughtWorks
Testing

ThoughtWorks
Done

ThoughtWorks
As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks
Ready

ThoughtWorks
In Dev

ThoughtWorks
Dev complete

ThoughtWorks
Testing

ThoughtWorks
Done

ThoughtWorks
Rendering bug

ThoughtWorks
As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

DEFECTS MANAGEMENT

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks

Rendering bug

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks

Rendering bug

Should this

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

ThoughtWorks

As a ... I want to ...
so that .

We will be done when

Given ... When ... Then ...

COMMON TESTING ISSUES / SMELLS

Is it really a defect?

Bouncing defects

Automation failures

Accepting stories, then raising defects

Business logic in tests

QUESTIONS?