The Agile Modules
CSC3045 & CSC3052

eXtreme Programming

School of Electronics, Electrical Engineering & Computer Science

Queen's University, Belfast

Dr Darryl Stewart

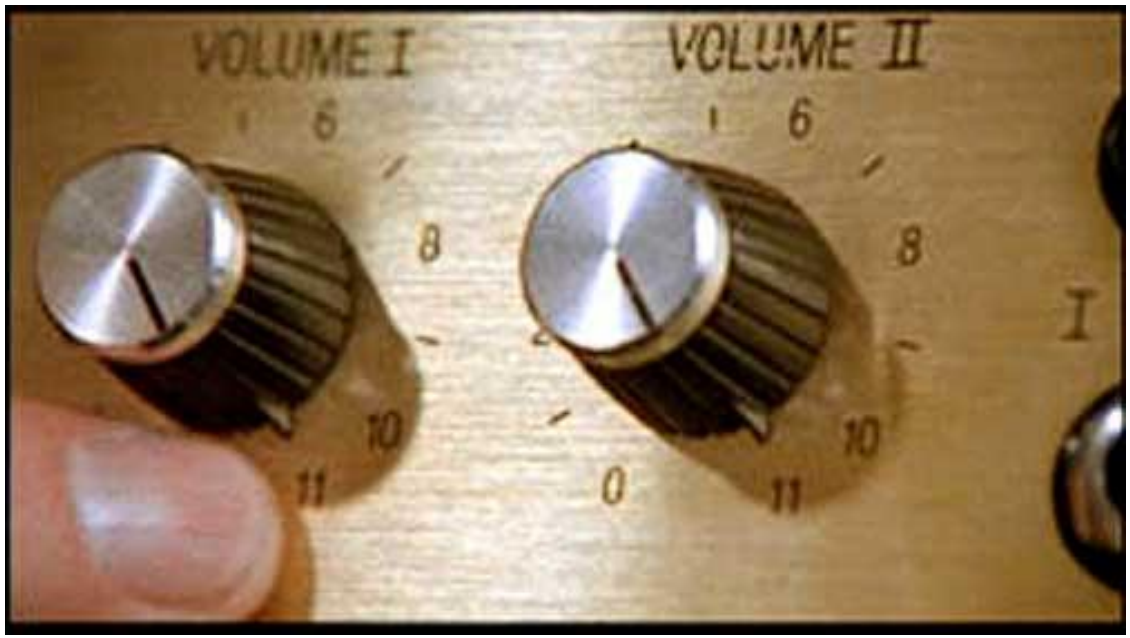
What is eXtreme Programming (XP)?

- XP is one of the most popular forms of Agile Development
- Pioneered by Kent Beck, along with Ward Cunningham and Ron Jeffries during a project for Chrysler in 1996
- Goal of XP is outstanding software development at lower cost, with fewer defects, high productivity and much higher return on investment

What it is NOT

What is XP?

- Provides a set of highly effective Practices for Software development
- Not only Practices but a philosophy of software development based on well considered Values and Principles
- Strong emphasis on small iterations, simple design, pair programming, test driven development and ongoing user involvement
- Gets its name from the fact that it takes industry best practices to their logical extremes “turns the knobs up to 10” - (Kent Beck)



What is XP?

XP addresses risks at all levels of development process:

- **Schedule Slips** ——— through short release cycles
- **Defect Rate** ——— by writing tests function-by-function
- **Business misunderstood** – customer is part of the team
- **Business changes** ——— through short release cycles
- **False feature rich** ——— only highest priorities are addressed
- **Staff turnover** ——— through staff empowerment

Five Core Values of XP

Fourteen Principles of XP

- **Humanity**
- **Economics**
- **Mutual Benefit**
- **Self Similarity**
- **Improvement**
- **Diversity**
- **Reflection**
- **Flow**
- **Opportunities**
- **Redundancy**
- **Failure**
- **Quality**
- **Baby Steps**
- **Accepted Responsibility**

XP **Practices**

Can be separated into:

- **Primary Practices**
  - Are useful independently
- **Secondary Practices**
  - Should be adopted after mastering the Primary Practices
  - Will amplify the benefits of Primary Practices

XP **Primary Practices**

In no particular order:

- Sit Together
- Whole Team
- Informative Workspace
- Energised Work
- Pair Programming
- User Stories
- Weekly Cycle

- Quarterly Cycle
- Slack
- Ten Minute Build
- Continuous Integration
- Test-first Programming
- Incremental Design

Informative Workspace - Primary Practice

- Display information about the project status all over in the workspace
- Place user story cards or big charts for issues that require steady progress
- Put coloured task post-its on a progress board
- Have lots of white boards for diagrams and brainstorming of architecture
- Have a clear build status indicator

Whole Team - Primary Practice

- Create a cross-functional team, with all the skills and perspectives needed for the project to succeed
- People should have a sense of “team”:
  - We belong
  - We are in this together
  - We support each others’ work, growth, and learning
- Team size is not restricted but less than 16 is usual (Scrum says 3 to 9)
- Avoid fractional team members
- Team can change as the requirements of the project change

Sit Together - Primary Practice

- Develop in an open space big enough for the whole team
- Stimulates information exchange by making it simple to just turn to the colleague and ask
- Requires a significant level of trust and alignment of the work
  - Less irritating to be disrupted by a side talk on a relevant topic, than by something you have no interest in
  - Both trust and work alignment are usually developed in the course of agile software development and may need to be developed before this practice is adopted
- Often private cubicles or offices are also available around the open space for phone calls or private meetings etc.

Microsoft Patterns and Practices Labs - Seattle

Pair Programming - Primary Practice

- Write all production code with two people sitting at one machine

User Stories - Primary Practice

- User Story = customer-visible functionality provided by a system
- Build a list of stories based on discussions with customers
  - Just note down the names initially
- Should be on the story wall for everyone to look at
- Role of stories:
  - Estimation
  - Units of planning and monitoring a system
  - Prioritization by customers
  - Risk assessment and estimation by developers

Energized Work - Primary Practice

- Work only as many hours as
  - you can be productive
  - you can sustain
- Avoid using up tomorrows energy today
- Ideally 40 hours a week
- Programming productivity is seen to be 4-5 hours a day
- Software development requires insight – you need to have a rested and relaxed mind
- Don’t work when sick!!!!!! - doesn’t help team
- Try things like ‘Code Time’ to increase productivity – phones are off hook and email closed
- Projects that require overtime to finish will finish late – no matter what
- Overtime is a symptom of problems in the project

avoid this

- **Try the POMODORO technique to**

Weekly and Quarterly Cycle - Primary Practice

- Plan at the start of each week and quarter

- Stories for the week selected by customer
- Progress made so far is assessed
- Similar to a one week Scrum sprint

- Pick a theme for the stories for the next quarter
- Address any bottlenecks in the process
- Initiate repairs

Slack - Primary Practice

- Important to establish and maintain the credibility of the release plans.
- It is better to under commit and deliver what is committed to rather than over commit and under deliver
- Introduce slack in your release plan, but not slyly
- Slack = including some low value, non-critical stories in the release plan that can be dropped if the schedule starts slipping

Ten Minute Build - Primary Practice

- Try to ensure that the project build takes no longer than 10 minutes
- Automate the build if possible
- Encourages people to actually do the build continuously
- Reduces thumb twiddling time

Continuous Integration - Primary Practice

- Integrate and test changes after no more than a couple of hours
- Integration can take longer than the original coding
- *For your project – commit your code to the repository every 30 minutes (only if it compiles and passes tests of course)*

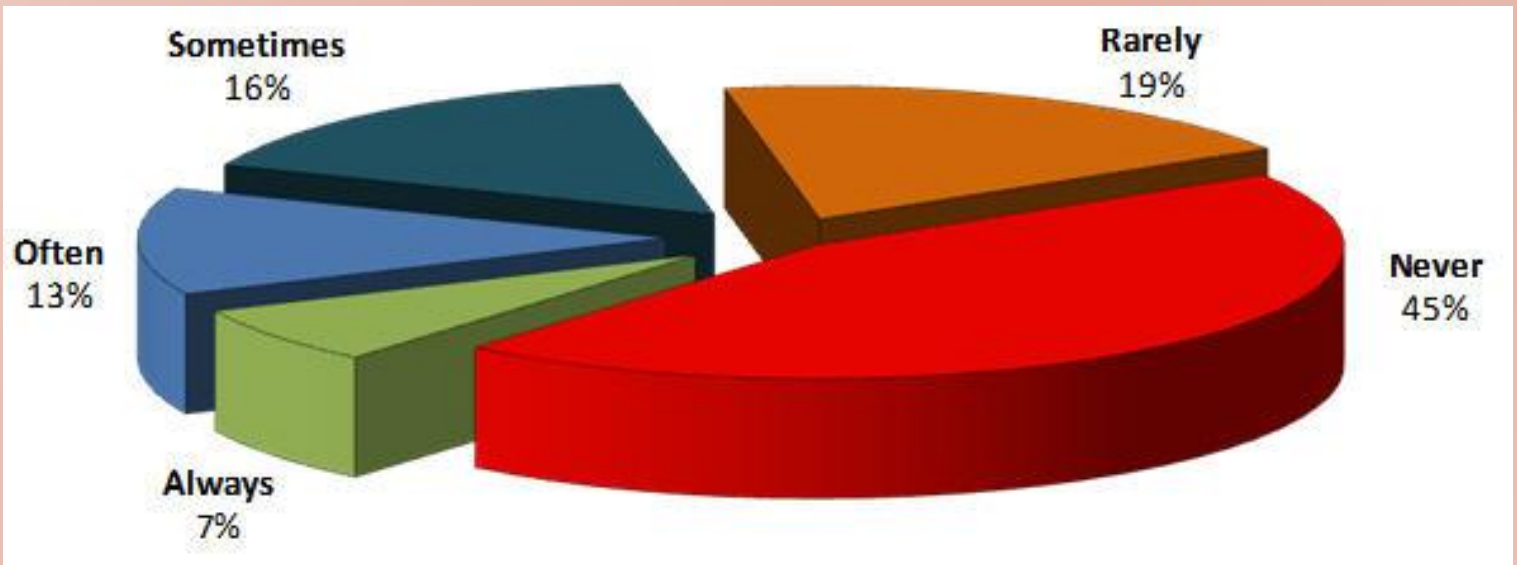
Test First Programming - Primary Practice

AKA: Test Driven Development

- Write tests for all code **before** you code
- Automate the tests using tools such as ***JUnit, NUnit***, etc.
- Code is completed only when it passes all tests
- Maintain the tests along with your code
- Write code at various levels: unit tests, acceptance tests, etc.

Incremental Design - Primary Practice

- Avoid “complete design before implementation”
- According to a ***Standish Group*** report, features and functions are used:



- Design what is needed <u>now</u>
- Keep investing in the design every day
- Create Spike solutions to tackle tough technical or design problems
- Design done close to when it is needed is more efficient
- **Refactor** your design as you go ahead – aided by automated tests

Secondary (Corollary) Practices

- **Real Customer Involvement**
- **Incremental deployment** – avoid big high risk switchovers
- **Team Continuity**
- **Shrinking Teams** – strive to reduce the load for one of the team
- **Root Cause Analysis** – fix a defect and then ask 5 whys?
- **Shared Code** – anyone in the team can improve any part of the system
- **Code and Tests** – maintain only the code and tests and minimise all other artefacts
- **Single Code Base** – minimise the number of different versions of the code that are being developed
- **Daily Deployment**
- **Negotiated Scope Contract** – reduced risk by signing a sequence of short contracts instead of one long one
- **Pay-Per-Use** – feedback on relevant features for future development

Core Value - **Communication**

- Problems in development often caused by
  - Miscommunication
  - Lack of communication
  - Lack of knowledge
- When a problem is encountered, ask
  - Was this due to lack of communication?
  - What communication would now address this problem?
  - What communication would avoid this problem future?
- The working environment can have a large effect on the quality of team communication

Core Value - **Simplicity**

- Striving for simplicity is the biggest challenge in XP
- The XP mantra is “Do the simplest thing that could possibly work” not just “Do the simplest thing”
- Be on guard against ‘contrived complexity’
- Make the system simple enough to “gracefully” solve only today’s problem
- Improved communication can aid in this quest for simplicity
  - Eliminating unneeded requirements
  - Clarifying usage scenarios

“Simple is efficient”

Core Value - **Feedback**

- Key aspect of all Agile development methods
- Change is inevitable – sooner realised the better
- XP encourages shortening the feedback cycle as much as possible – to minutes or hours instead of weeks or months
- Forms of feedback:
  - Opinions about ideas
  - How a solution looks when it is implemented
  - Whether the tests were easy to write – gives feedback on simplicity
  - Whether the tests run – gives feedback on correctness
  - How an idea works after implementation – yours, teams, customers, users
- Feedback is part of Communication and contributes to Simplicity

Core Value - **Courage**

- Defined as “effective action in the face of fear”
- Sources of fear in this context could be :
  - The responsibilities taken on as part of a team
  - Exposing programming frailties during pair programming
  - Facing difficult problems with as yet unclear solutions
  - Realisation of mistakes made or sub-prime solutions previously developed
  - The “known unknown” complexity ahead
- Courage in this context would then mean
  - Tell the truth, pleasant or unpleasant – good **Communication**
  - Discard failing solutions and seek new ones – in search of **Simplicity**
  - Ask questions to get concrete answers – creates **Feedback**
    - Adopt a “there are no stupid questions” approach
  - Be aware that everything is difficult when you are first learning it or solving it

Core Value - **Respect**

- If members of a team don't care about each other – **XP won't work**
- If team members don't care about their work – **XP won't work**
- If team members don't care about a project – **XP will not make it work**

"everyone has something to contribute — it's easy and tempting, when someone ticks you off or is mistaken (or both), to simply disregard all their input in the future by setting the "bozo flag" to TRUE for that person. But by taking that lazy way out, you poison team interactions and cannot avail yourself of help from the "bozo" ever again." taken from *Dynamics of Software Development, by* Jim McCarthy - Rule #4

"Don't Flip the Bozo Bit"

XP Principles

These should be considered and balanced when making decisions:

- **Humanity** - What do people need to become good developers?
- **Economics** - Every action should have business value
- **Mutual Benefit** - Most important and most difficult to adhere to. Extensive internal documentation benefits who?
- **Self Similarity** - You can copy structure of one solution to a new context. Look for patterns in themes, stories, tests

- **Improvement** - In software development “perfect” is a verb not adjective. The software cannot **be** perfect but you can continually strive **to** perfect it
- **Diversity** - Teams need diversity
- **Reflection** - How and Why of working
- **Flow** - Steady flow of valuable software
- **Opportunities** - Problems are opportunities to improve
- **Redundancy** - Do not remove redundancy that serves a valid purpose (multiple practices to catch bugs)

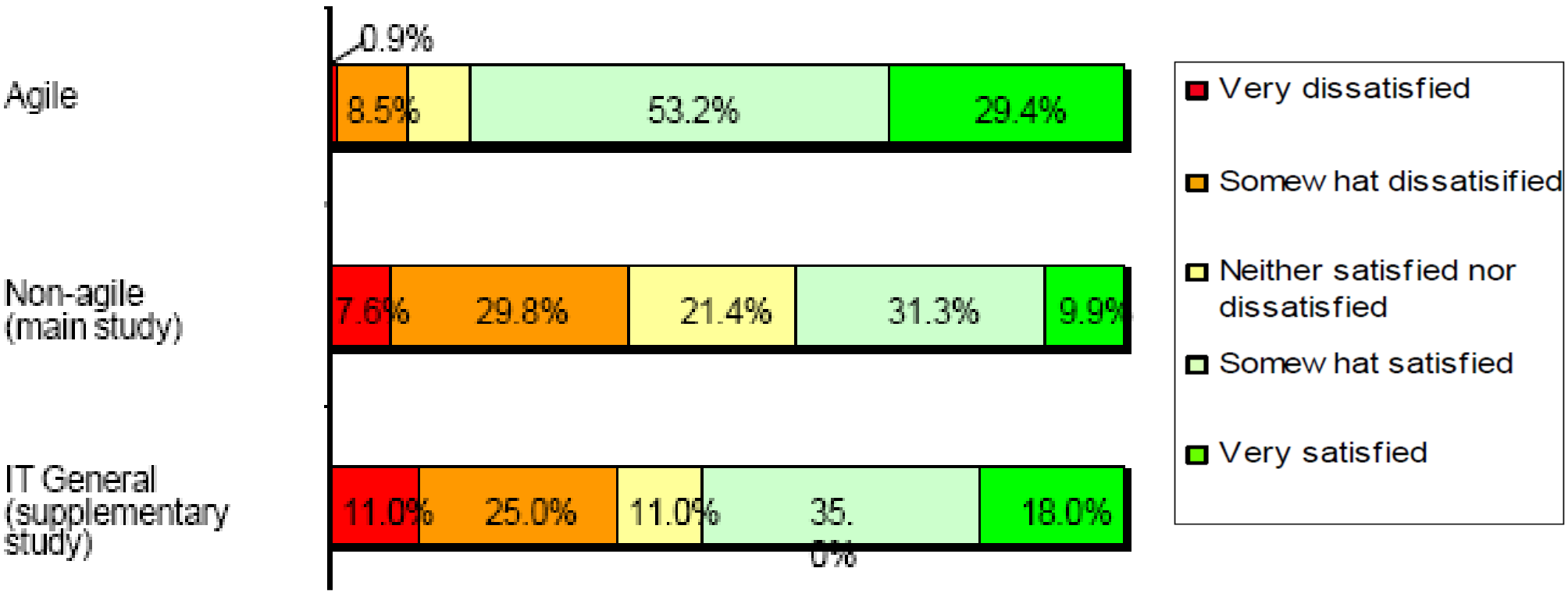
XP Principles

- **Failure** - Is failure a waste?
- **Quality** - Cost of quality? Quality ~ Productivity (reduced quality may increase productivity in the short term but will decrease productivity much more over the long term)
- **Baby Steps** - Rapid small steps = leap, (each step is low risk)
- **Accepted Responsibility** - Responsibility cannot be assigned, it can only be accepted

Our survey said...

Grigori Melnik, Frank Maurer, "Comparative Analysis of Job Satisfaction in Agile and Non-agile Software Development Teams", Extreme Programming and Agile Processes in Software Engineering, 2006

- Overall job satisfaction by groups



Take home messages

- eXtreme Programming – not what it sounds like!
- Not **all** about the code
- Based on an explicit **value system**
- Follows a broad range of **principles** focused on effective software engineering
- Describes a broad range of actual things **you can do** (practices) that will improve the process and the outcome of your software development
- Is a flexible approach which can be adopted in part or in full depending on available resources/expertise/experience etc.