University of Dublin
Trinity College

CS7CS3: Introduction to eXtreme Programming

Prof. Siobhán Clarke
Ext. 2224 – L2.15
www.scss.tcd.ie/Siobhan.Clarke/

The Session Will Begin Shortly

1. Click on the pink Collaborate button (bottom right) to open the chat window and enter your message.

2. You can close the Collaborate panel at any time so you can see more of the current presentation.

3. Click on the menu icon at the top left when you want to exit the online lecture

→ A recording will be made available afterwards in case you get disconnected or have technical issues.

eXtreme Programming

So called “Agile Software Development” – XP emerged around 2000

Individuals and interactions *over* processes and tools

Working software *over* comprehensive documentation

Customer collaboration *over* contract negotiation

Responding to change *over* following a plan

.. Idealistic view of how to develop software that may not be appropriate in every situation.

.. Lots of good ideas that can be applied to other processes.

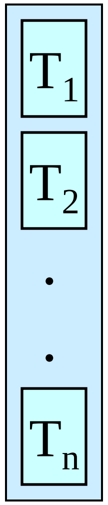
.. Very successful in many small-medium projects

eXtreme Programming - 2

Define features of system $\longrightarrow$ Client informed cost of each (time/money)

For each successive "build" of system:

Client selects features — Broken into tasks $\rightarrow$



In parallel:
pairs of programmers:
define test cases
implement
integrate

Computers of XP team set up in centre of large room
Client representative works with the team
No individual can work overtime for two successive weeks
No specialization – all members specify, design, code, test
Design modified while product is built – refactoring

eXtreme Programming - 3

- Stories
  - *A software feature from an end user’s perspective*
  - *Drive acceptance testing*
  - *Some number of stories implemented in each iteration*
  - *Only enough detail for estimation purposes*
  - *Detailed requirements specification done with user during iteration*
- Sprint vs Iteration
  - *Sprint: unit of work that can be delivered to client – a product increment*
  - *Iteration: may still be subject to change, so not necessarily a product increment*
- Velocity
  - *A measure of work achieved in a “sprint”*
  - *Importantly – based on estimates!*
  - *Can be useful over a number of iterations with the same team*
  - *Lots of tools to measure velocity*

eXtreme Programming – 12 practices

Fine scale feedback
- Planning Game
- Pair Programming
- Test-Driven Development
- Whole Team

Continuous process
- Design Improvement
- Customer Tests
- Continuous Integration

Shared understanding
- Simple Design
- Collective Code Ownership
- Coding Standards
- Metaphor

Programmer welfare
- Sustainable Pace

Many of these can be applied to other processes…

Whole Team

"All the contributors to an XP project – developers, business analysts, testers, etc. – work together in an open space, members of one team. The walls of this space are littered with big visible charts and other evidences of their progress"

Whole Team

"All the contributors to an XP project – developers, business analysts, testers, etc. – work together in an open space, members of one team. The walls of this space are littered with big visible charts and other evidences of their progress"

Comments from previous years:

Benefits:	Drawbacks:
• all knew who was responsible for what • everyone a question away • helped with regular integration • open forum for discussion	• difficult to concentrate in small space • sometimes too much time spent discussing problems as a group

Planning Game

“Planning is continuous and progressive. Every two weeks, for the next two weeks, developers estimate the cost of candidate features, and customers select those features to be implemented based upon cost and business value”

Planning Game

"Planning is continuous and progressive. Every two weeks, for the next two weeks, developers estimate the cost of candidate features, and customers select those features to be implemented based upon cost and business value"

Comments from previous years:

Benefits:	Drawbacks:
• very beneficial – higher priorities dealt with first – everyone with the same view of high priorities • helped maintain focus – curbed wild goose chases	• with short iterations, was difficult to meet client every time • difficult to get estimation right • found that time was wasted waiting for next iteration to start

Customer tests

"As part of selecting each desired feature, the customers define automated acceptance tests to show that this feature is working "

Customer tests

"As part of selecting each desired feature, the customers define automated acceptance tests to show that this feature is working "

Comments from previous years:
For MSc group projects, customer access was too difficult.

Simple design

“The team keeps the design exactly suited for the current functionality of the system. It passes all the tests, contains no duplication, expresses everything the authors want expressed, and contains as little code as possible ”

Simple design

"The team keeps the design exactly suited for the current functionality of the system. It passes all the tests, contains no duplication, expresses everything the authors want expressed, and contains as little code as possible "

Comments from previous years:

Benefits:	Drawbacks:
• did not waste time embellishing software with unimportant functionality • curbed design overkill • had simple working system at earliest possible point	• None! • May encourage avoiding issues that may come back to haunt you.

Pair programming

"All production software is built by two programmers, sitting side by side, at the same machine"

Pair programming

"All production software is built by two programmers, sitting side by side, at the same machine"

Comments from previous years:

Benefits:	Drawbacks:
• We all knew who was doing what at all times	• Cultural change very hard
• Knowledge was spread among the team.	• Learning curve
• stronger sense of responsibility in the team	• Frustrating at the start
• two minds better than one – learned a lot from each other	• A team member on their own could have done some very simple tasks – other member of pair felt redundant.
• helped nail down conventions	• *(important to swap pairs)*

Test-driven development

“The programmers work in very short cycles, adding a failing test, then making it work”

Test-driven development

"The programmers work in very short cycles, adding a failing test, then making it work"

Comments from previous years:

Benefits:	Drawbacks:
• Can verify whole system at any time, very easily • Benefits of a test framework are uncountable! • High level of confidence of quality • Rarely felt bogged down with the exact same problem for a long time. • Always had "previous" working version – psychologically good!	• Takes a while to get used to (but worth it!) • Big learning curve • A lot of time designing tests • Should make tests independent of other tests, or of having specific data in the system

Design improvement

“Don’t let the sun set on bad code. Keep the code as clean and expressive as possible “

Design improvement

"Don't let the sun set on bad code. Keep the code as clean and expressive as possible "

Comments from previous years:

Benefits:	Drawbacks:
• Bad code and design immediately removed – doesn't hang around to haunt you • Keeping design simple makes it easier to modify and maintain • Easier for people who hadn't worked on code to understand it	• Time consuming to refactor for the sake of refactoring – tried to choose just to simplify what working on now. • No drawbacks! • None!

Continuous integration

“The team keeps the system fully integrated at all times ”

Continuous integration

"The team keeps the system fully integrated at all times "

Comments from previous years:

Benefits:	Drawbacks:
• Always had a working system to baseline against.	• Initial problems with CVS
• Problems revealed earlier	• No drawbacks!
• Made overall process simpler	• None!

Collective code ownership

‟Any pair of programmers can improve any code at any time ‟

Collective code ownership

"Any pair of programmers can improve any code at any time "

Comments from previous years:

Benefits:	Drawbacks:
• Great! – becomes a team effort • code no longer a monkey on your back! • increased communication and reduced confusion • "no blame" relieved a lot of pressure	• Initially hard to get used to – good when you get to the point where you don't know who wrote the code • people naturally gravitate towards code they're more interested in • None!

Coding standard

‟All the code in the systems looks as if it was written by a single – very competent – individual‟

Coding standard

"All the code in the systems looks as if it was written by a single – very competent – individual"

Comments from previous years:

Benefits:	Drawbacks:
• easier to manage code base • helps when moving from module to module • makes refactoring a lot easier	• lingering dissatisfaction if standard used not your own • widespread slips could have caused problems • difficult to change your own style

Metaphor

“The team develops a common vision of how the program works ”

Metaphor

"The team develops a common vision of how the program works "

Comments from previous years:

Benefits:	Drawbacks:
• Everyone had a similar understanding of the system – easier to communicate • Shared vision of "big picture"	• Created an onus that everyone understand the whole system – at times an unnecessary burden

Sustainable pace

“The team is in it for the long term. They work hard, at a pace that can be sustained indefinitely. They conserve their energy, treating the project as a marathon rather than a sprint ”

Sustainable pace

"The team is in it for the long term. They work hard, at a pace that can be sustained indefinitely. They conserve their energy, treating the project as a marathon rather than a sprint "

Comments from previous years:

Benefits:	Drawbacks:
• Felt fresh starting each session – didn't feel jaded or bored • With other assignments, nice to have planned time for this project • Know exactly how much time you have to get the work done	• When team got used to a fixed schedule, was very difficult to schedule more time when it was required • Hard to define timetable because of mixed schedules

Core values

Simplicity,

Communication,

Feedback and

Courage

Core Values - Simplicity

- Delivering the simplest functionality that meets business needs

- Designing the simplest software that supports the needed functionality

- Building for today and not for tomorrow

-Writing code that is easy to read, understand, maintain and modify

[critics: could result in more re-design effort in the long run]

(Extreme Programming White Paper: Thomas Meloche, Menlo Institute)

Core Values - Communication

- Collaborative workspaces
- Co-location of development and business space
- Paired development
- Frequently changing pair partners
- Frequently changing assignments
- Public status displays
- Short standup meetings
- Unit tests, demos and oral communication, not documentation

Core Values - Feedback

- Aggressive iterative and incremental releases
- Frequent releases to end users
- Co-location with end users
- Automated unit tests
- Automated functional tests

Core Values - Courage

- Do the right thing in the face of opposition
- Do the practices required to succeed

Roles

Programmer

In addition to usual know “how to”: good communicator (pair programming); habit of simplicity; refactoring skills; ability to acknowledge fears!

Customer

Knows “what to”; Has to actually make decisions! Learn to write stories and functional tests

Tester

Testing really responsibility of programmers, but tester role focuses on helping customer choose and write functional tests. (May not be a separate person)

Tracker

“Conscience” of the team (e.g., “last time(s), estimates were wrong”); Keeps eye on big picture; Team historian – log of tests, defects reported and new resulting tests.

Coach

Responsible for team as a whole; Knows process well. Remain calm in panic!

Consultant

Paid to be there for a short time to solve a problem; Team questions/watches and then no longer needs him.

Big Boss

Needs to be supportive!

Impediments

- Biggest barrier is culture
  - if there’s a higher level management team that is insisting on up-front design, and loads of documentation – XP will be difficult
  - Culture of long hours to prove commitment hard to reconcile
- Team size makes a difference
  - Doable with 100?, 50?, 20?…. 10 definitely doable
- Technical environment
  - Can’t keep solution simple because of external constraints (e.g., a database that has to be used and you have to change without breaking other apps)
  - If compile/link takes 24 hours, XP won’t work
- Physical environment
  - Programmers should not be physically separated
- Change management
  - Unstable requirements, user conflicts, revision funding

More detail…

www.extremeprogramming.org

(as with all things – lots of comments for and against on the web)

What about SCRUM?

- Focus on management of project with "SCRUM" meetings:
  - Daily Scrum
  - Sprint Planning Meeting
  - Sprint Review Meeting
  - Sprint Retrospective

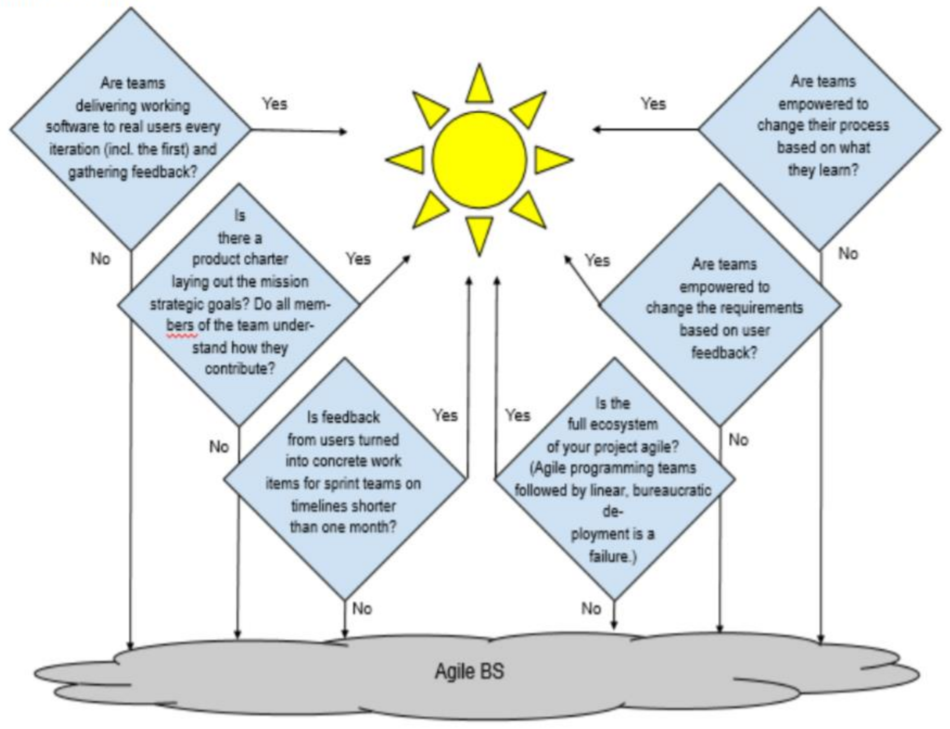
•Differences between SCRUM and XP

http://blog.mountaingoatsoftware.com/differences-between-scrum-and-extreme-programming

- SCRUM: "just XP without the Engineering Practices"??

Claiming to be agile?

Graphical version:


Are teams delivering working software to real users every iteration (incl. the first) and gathering feedback?
Yes
No
Is there a product charter laying out the mission strategic goals? Do all members of the team understand how they contribute?
Yes
No
Is feedback from users turned into concrete work items for sprint teams on timelines shorter than one month?
Yes
No
Are teams empowered to change their process based on what they learn?
Yes
No
Are teams empowered to change the requirements based on user feedback?
Yes
No
Is the full ecosystem of your project agile? (Agile programming teams followed by linear, bureaucratic deployment is a failure.)
Yes
No
Agile BS

More information on some of the features of DoD software programs are included in Appendix A (DIB Ten Commandments on Software), Appendix B (DIB Metrics for Software Development), and Appendix C (DIB Do's and Don'ts of Software [to be linked]).

https://media.defense.gov/2018/Oct/09/2002049591/-1/-1/0/DIB_DETECTING_AGILE_BS_2018.10.05.PDF