CSC3031 This Week

- This is Week 3. If you still haven't been in contact with your supervisor, **now** is the time to do it.
- Assignment deadlines moved as we announced on 8th Feb. Next submission is 25th February for your assessed presentation recording, and your non-assessed participation in peer review. Project proposal and ethics form due on 4th March.
- Peer review is strongly encouraged – please participate in your small online group. see our announcement in Canvas.
- This Week's Topics:
  - Software Development
  - Reading the CS Literature
  - Dealing with Qualitative Data
  - Elements of Technical Writing
  - Guest Lecture: Alexander Wilson on JigsAudio, designing slow technology for public consultations

CSC3031 Next Week

- Next week, all classes are drop-in workshops and surgeries, but we are very happy to arrange topic-focussed classes as well.
- **What topics would you find useful?** Let us know in the discussion that we have opened up on Canvas.
- **Need Help?** Your supervisor should be your first point of contact, but we continue to hold our tutorials for any general questions and support. Our online tutorials are still available –sign up in Canvas.

CSC3031 - Research and Project Skills

Project Skills: Aspects of Software Development for Stage 3 Projects

John Fitzgerald

Aspects of Software Development

Your project is likely to involve software development in some form. It’s worth considering early how you might organise your development work.

1. Software Development Life Cycle
2. From Processes to Stages: organising your software development
3. Software Quality and the Importance of Validation and Verification

Aspects of Software Development

Why bother with a defined software development life cycle?

- Better management of a complex task
- Makes progress visible
- Gives structure
- Better software and documentation

There are some standard models, but the key is to define the right process for <u>your</u> project.

Software Development Processes

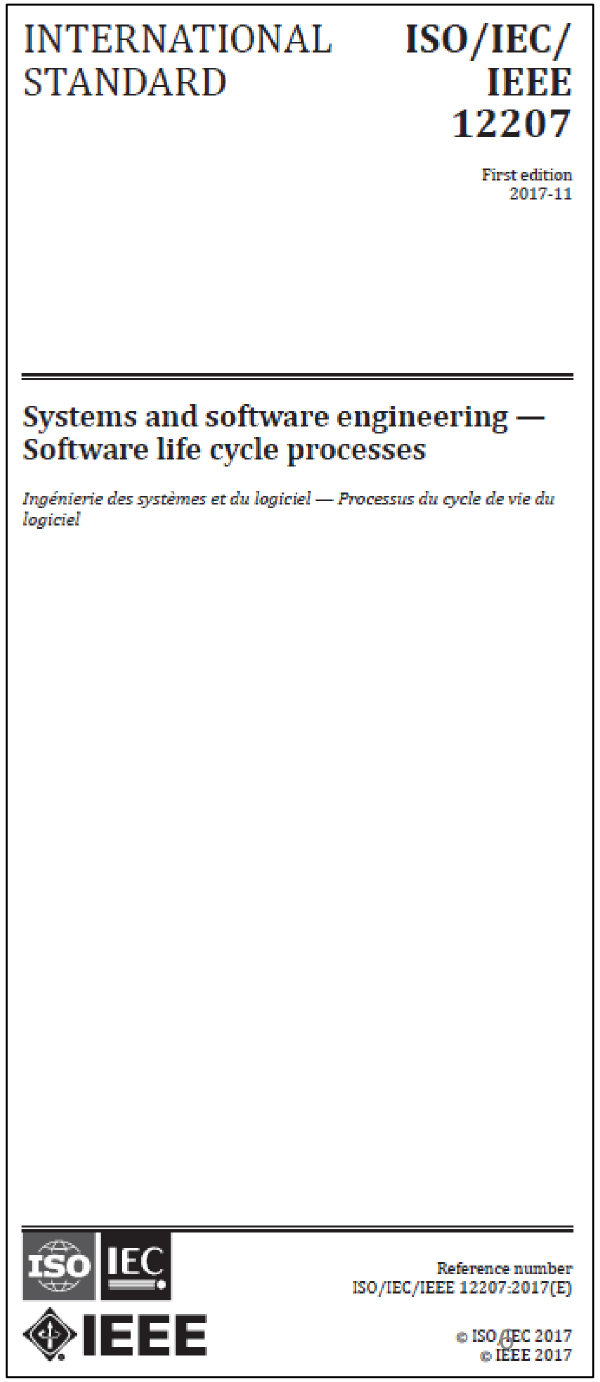
International Standard ISO/IEC/IEEE 12207 – 2017 defines processes, but not how they should be organised within a given project. It identifies 14 processes which cover:

Agreement Processes

Organisational Project-enabling Processes

Technical Management Processes

Technical Processes



INTERNATIONAL STANDARD

ISO/IEC/ IEEE 12207

First edition 2017-11

Systems and software engineering — Software life cycle processes

Ingénierie des systèmes et du logiciel — Processus du cycle de vie du logiciel

Reference number ISO/IEC/IEEE 12207:2017(E)

© ISO/IEC 2017
© IEEE 2017

Software Development Processes

Agreement Processes
Reaching agreement between supplier and acquirer, including project management plan.

Organisational Project-enabling Processes
Lifecycle, infrastructure, people, quality and knowledge management.

Technical Management Processes
Project control, decision making and risk management, measurement.

Technical Processes
Requirements definition, design, implementation, verification, maintenance, disposal.

Software Development Processes

Agreement Processes
Reaching agreement between supplier and acquirer, including project management plan.

Organisational Project-enabling Processes
Lifecycle, infrastructure, people, quality and knowledge management.

Technical Management Processes
Project control, decision making and risk management, measurement.

Even though we concentrate mostly on Technical processes, your project has some of the other processes going on in some measure.

Software Development Processes

- We will concentrate on Technical Processes, broadly relating to:
  - Requirements
  - Design
  - Implementation
  - Validation & Verification

On the right side of the following slides, we'll give tips for each process that could be useful in your project. They won't be right for all projects, though, so consult your supervisor.

Requirements Processes

- There's an extensive literature on requirements engineering
- The process of deriving specifications of computing systems from a client's expression of needs and desires.
- Moving from expression in terms of the client's domain of expertise to specifications that we can code.

Consider Functional and Non-functional Requirements:

- **Functional:** define what the product is required to do (including interfaces)
- **Non-functional:** constraints, standards, limitations including schedule.

Rank Requirements as 'MoSCoW':

- **Must have**
- **Should have**
- **Could have**
- **Won't have**

Use a cost/value analysis to do the ranking.

You might need interviews, surveys or focus groups, or you can **define *personas*,** i.e. imaginary representatives of the stakeholder group you want to reach. Characterise your user!

Catalogue and label requirements for reference and traceability.

Design Processes

- Provide sufficient information about the system and its elements to enable implementation consistent with the functional and non-functional requirements.
- Involves defining a software architecture as well as interfaces, data, algorithms.

Consider using (a subset of) an established notation such as UML or SysML:

- Helps to reduce ambiguity
- Can give you a basis for ensuring consistency, e.g. between definitions in data and functionality.
- Tools are available to support these notations, but may be overkill for your project – consult your supervisor.

Traceability: exceptionally well documented designs provide traceability from elements of the design to the requirements that they are intended to satisfy.

- e.g., "This data structure is selected in order to satisfy Reqts. 1.1, 2.1, 3.2"

Implementation Processes

- Coding system elements in accordance with your design
- Integration of system elements (code units).
- Normally conducted alongside unit and integration testing (Validation & Verification processes).

Try to adhere to a discipline for configuration or version control:

- Make it lightweight but do record what versions you have produced, what has changed, and maintain a backup (via Git, SVN, etc.)
- Even issue version numbers and your own milestone releases.
- Seems like a pain, but as your implementation gets more complex, it really can be worth it.

Validation & Verification

- Can be an area of weakness in Stage 3 projects.
- Demonstrating that the software satisfies:
  - The design and detailed spec. (Verification)
  - The client's needs and wishes (Validation).
- Testing is a major tool in doing this demonstrating

Three ways to deal with defects in your product:

- **Avoidance:** prevent them in the first place by including regular V&V in each design stage
- **Detection & Removal:** test early, test often
- **Tolerance:** understand how good your product needs to be, e.g.,
  - Is it going to a real client (with your name on it for ever)?
  - Is it an experimental prototype for which showing feasibility is enough?
  - What balance do you need between functionality and quality? You may choose to deliver less functionality at higher quality.

Validation & Verification

- There's a wide range of testing activities applicable to different levels, e.g. Unit, Integration, System, Acceptance Test.
- Regression Test shows after a major change that your system still works (because fixing one defect can reveal another).

Incorporate a Test Plan in your project.

- What aspects of your product will you test?
- What test inputs or scenarios will you run?
- What results will be 'good enough'?
- Show the actual results and any action taken.

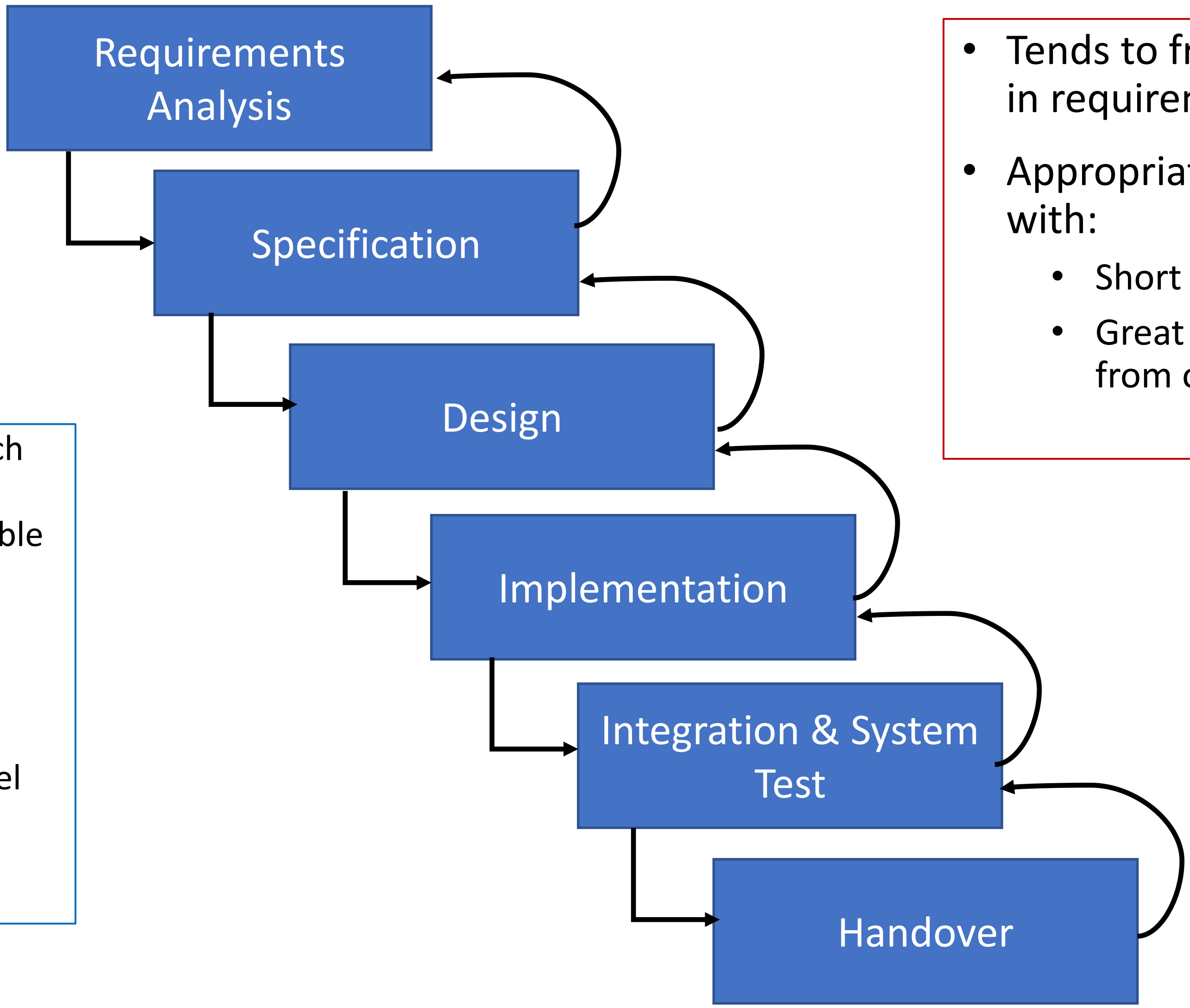
Strategies for selecting tests include:

- Designing a test set for each requirement (helps traceability – and you can do this before implementation)
- Executing a random selection of inputs
- Concentrating on boundaries (extreme values where defects often happen)
- Designing representative scenarios (input sequences).
- Constructing test cases to exercise particular areas of code that you believe are likely to include defects.

From Processes to Stages

- An early approach to software development was ‘build and fix’, but this becomes unsustainable as software scale and complexity grow.
- How will you structure Software Development processes to form a project lifecycle?
- Several models are available:
  - **Sequential models** structure processes into defined stages that produce outputs and follow one another in a straight execution path.
  - **Iterative models** have a series of development phases that repeat, gradually adding more functionality or other features until a sufficiently good product is available for handover.

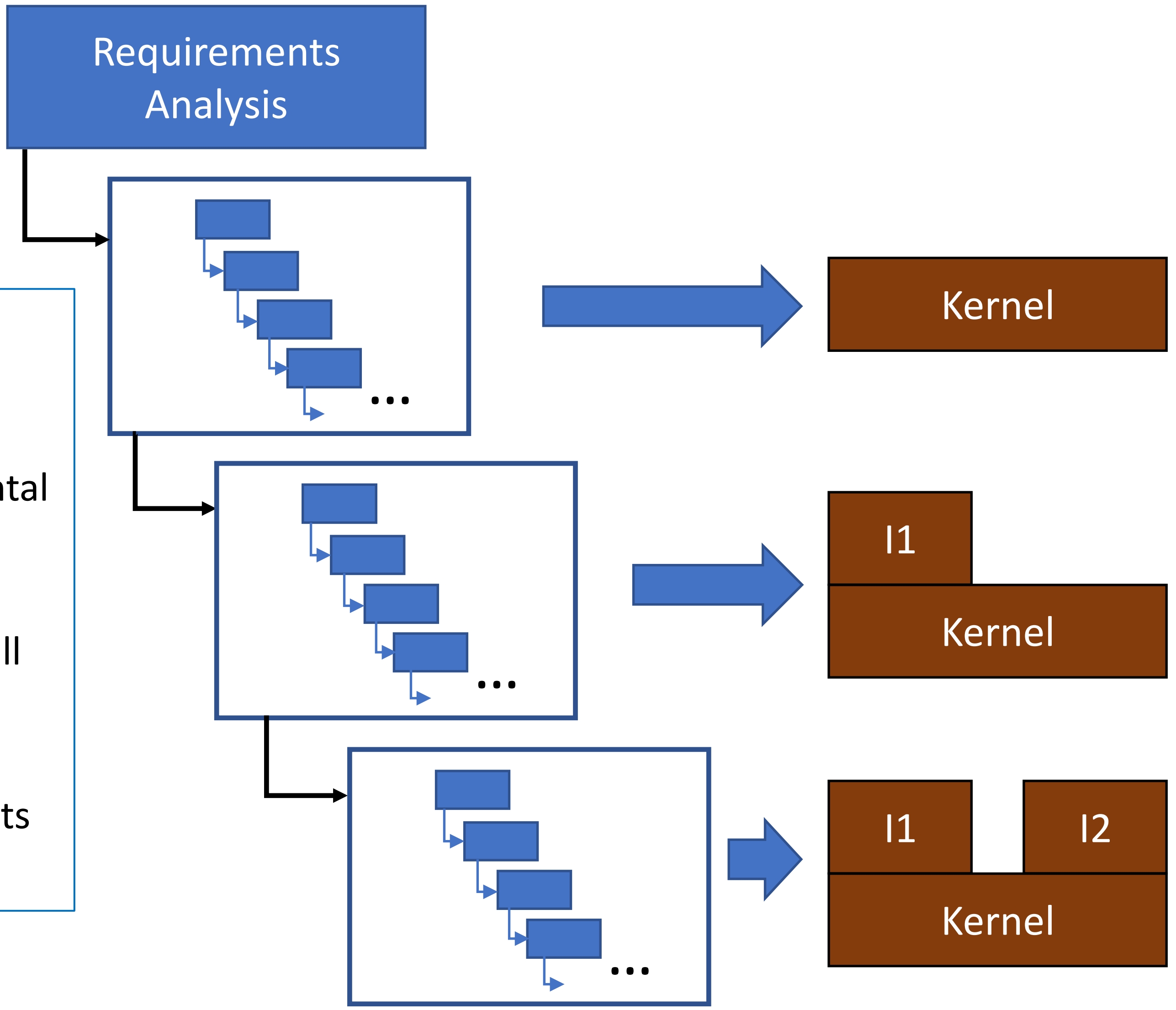
Sequential models: waterfall



- Tends to freeze out any changes in requirements.
- Appropriate only for projects with:
  - Short timeframe
  - Great clarity on requirements from outset

- Waterfall model completes each process before the next.
- Each stage produces a deliverable before the next to mange communication between development teams.
- Unit Testing in Implementation phase.
- See also the "V" Lifecycle model popular where verification is important.

Sequential models: incremental



- Develop a first small functional Kernel – get something working early
- Progressively add Incremental units on the basis of value/cost ratio.
- You need to have the overall architecture fixed from the outset
- Realistically, ~2-3 increments in a Stage 3 project.

Prototyping

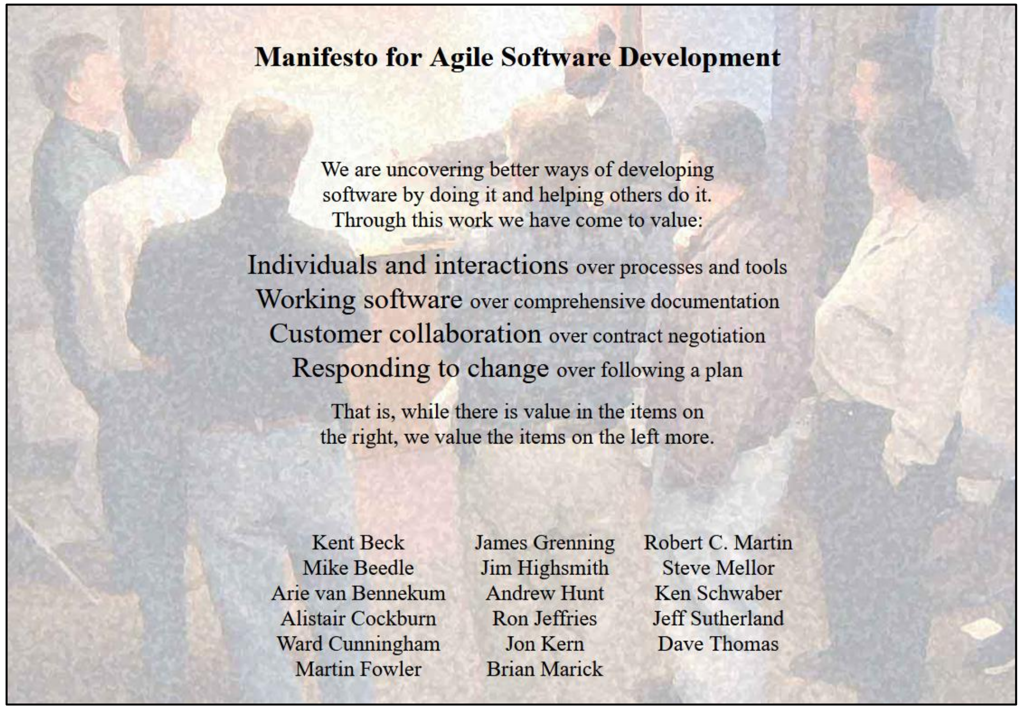
- Prototyping is an appropriate model where the requirements are not so well defined at the outset.
- You can use a prototype to
  - Explore or clarify requirements
  - Try out concepts, e.g. for interface design
  - Explore feasibility where the project has a lot of unknowns
- What do you do with the prototype?

Throw-Away Prototyping
- Lessons learned from the prototype are recorded.
- The prototype itself is discarded
- Begin a new development, e.g. in the intended language
- It’s often a wrench to throw a prototype away.

Evolutionary Prototyping:
- Start development from the initial prototype.
- Follow a series of mini-developments (incremental style), but improving the prototype at each version.
  - Add functionality or improve the rough prototype code at each version.

Agile Methods

- Sequential methods have been characterised as 'heavyweight'
- In response, 'Agile' development is characterised by:
  - Delivery in short bursts or releases
  - Getting a (partially) working product to the client quickly
  - Small reactive development teams (minimise communications overheads)
  - Embrace changing requirements
  - Self-organising teams that reflect regularly on how they can improve



Manifesto for Agile Software Development

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

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

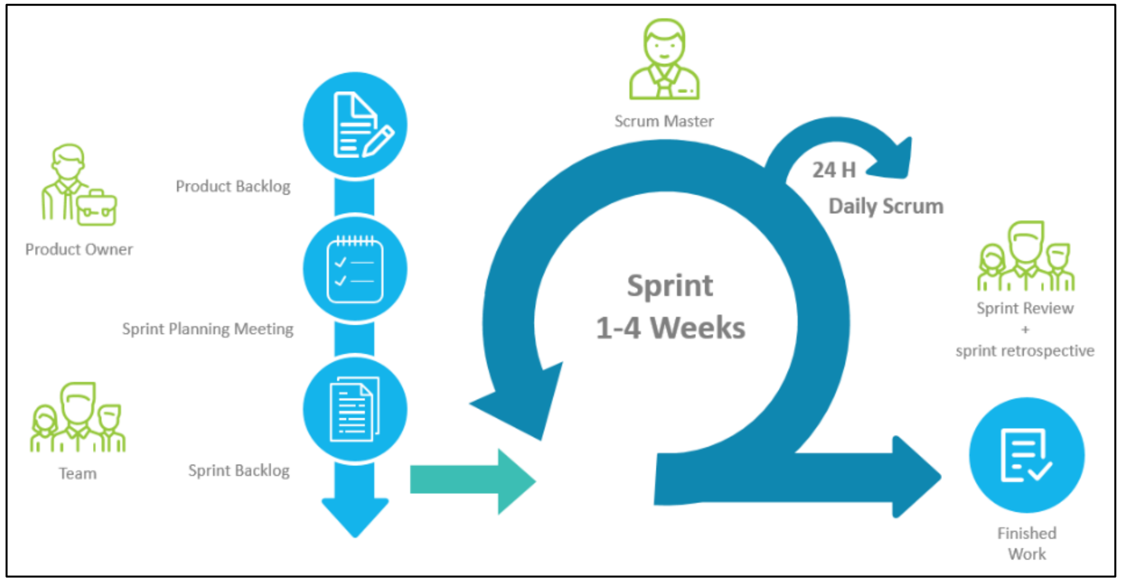
That is, while there is value in the items on the right, we value the items on the left more.

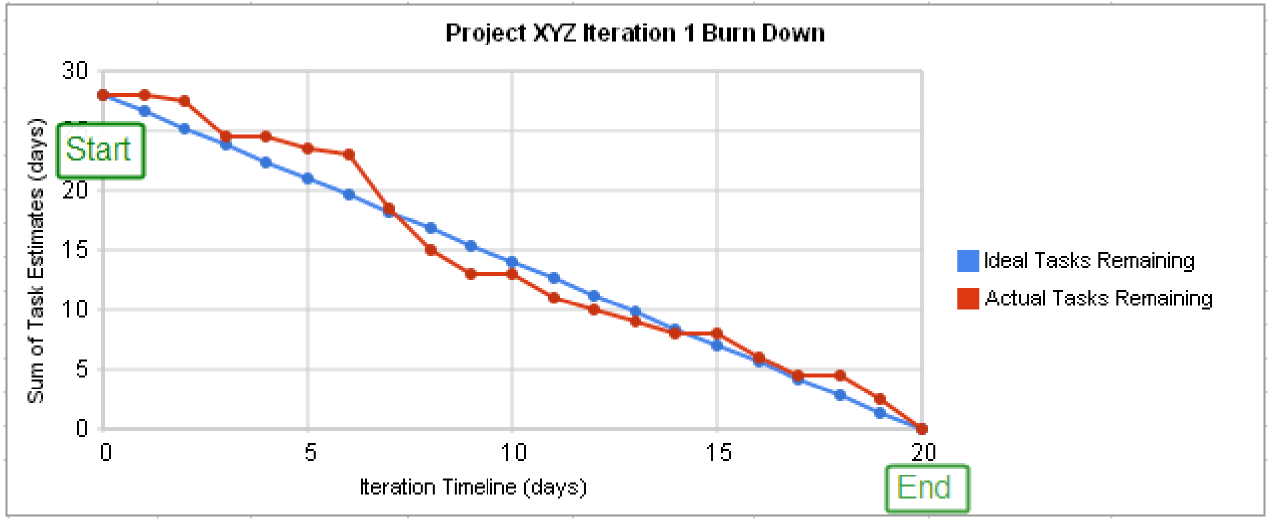
Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

https://agilemanifesto.org/ 2001

Agile Methods: Scrum

- Teams with defined roles (facilitator, product owner, team member)
- Work broken down into chunks addressed in sprints.
    - For example, choose a subset of requirements from a MoSCoW list and address them over a sprint period.
    - Maintain a visible record (e.g. graph) of remaining work to track progress.
- Emphasis on teams but the aspects of iteration, tracking and regular reflection could carry into your project.





What Should I use?

- In your final dissertation, you'll need to describe your development approach, and justify it, so discuss with your supervisor.
- My experience aligns with that of Dawson …

Approach	**Consider applying if …**
Waterfall-style	• Requirements understood well and unlikely to evolve. • Confident about implementation environment.
Incremental	• Complex software • Requirements clear • Solution architecture allows separation of elements into increments. • User can be reached to evaluate increments.
Throw-Away Prototyping	• Uncertain about requirements or feasibility of an implementation • Brief proof of concept is OK
Evolutionary Prototyping	• Uncertainties and risk in interface design or user requirements • Your programming environment is well understood.
Consider integrating aspects of agile methods with incremental and evolutionary approaches.	

CSC3031 - Research and Project Skills

Project Skills: Aspects of Software Development for Stage 3 Projects

John Fitzgerald