Software engineering

Computing & Information Sciences

W. H. Bell

University of Strathclyde Glasgow

Welcome to Module

Outline

University of Strathclyde Glasgow

- Introduce software engineering.
  - Concepts and practical applications.
  - Describing and delivering projects.
- Discuss real-life issues encountered.
  - Guest lectures from commercial software developers.

Assessment

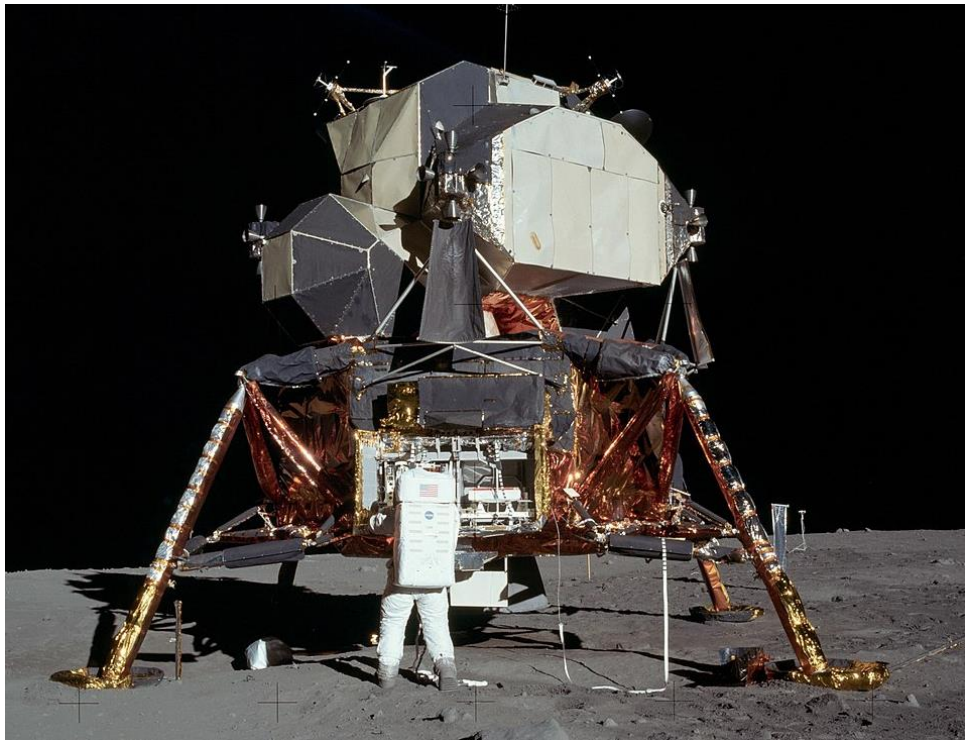
- Group project – 40%.
  - Produce technical document to describe software.
  - Assess commercial document writing.
  - Assess working in a team.
- Exam – 60%.
  - Multiple choice and written answers.

Software Engineering

Apollo guidance computer

Free flow ideas → Structured thought process

DSKY user interface



Apollo 11: Luna module

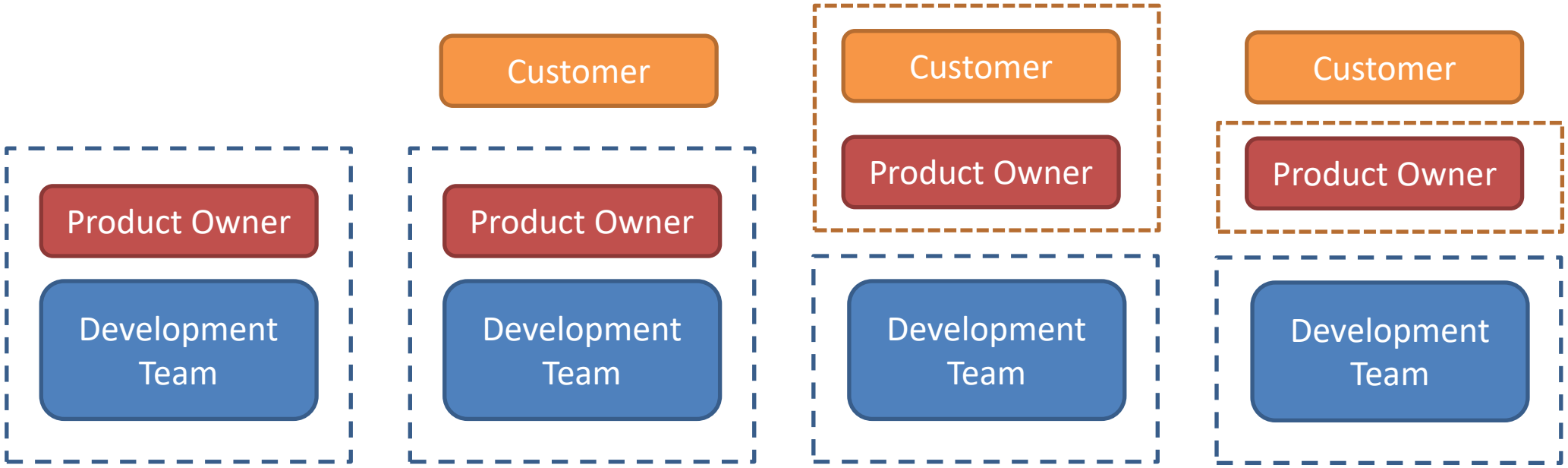
Margaret Hamilton, Lead Apollo guidance computer software engineer.

Apollo guidance computer

University of Strathclyde Glasgow

- Money was not an issue.
- Limited computational power.
  - Very small compared to modern systems.
  - Limited memory and storage.
- Limited software development time.
  - Needed to succeed quickly.
- Safety critical system.
  - Avoid death of astronauts.

Software development scenarios



Delivering internal services.

Company delivering applications or services.

Building software for a client.

Building software for a company that sells to others.

Objectives

University of Strathclyde Glasgow

- Need to satisfy stakeholders.
  - End users.
  - Company.
- Need to be paid.
  - Agreement of features that are present within a release.
- Minimal software defects.
  - Release often/less often – depends on application.

University of Strathclyde Glasgow

Teams and Risks

Development team

University of Strathclyde Glasgow

- One to many developers.
  - Lead developer.
  - Project manager – embedded or external.
- One or more test engineers.
  - Test framework.
  - Test bench – complex configuration.
  - Release platforms.

Information flow

- Describe what is being built.
  - Share architecture ideas with development team.
    - Internal structure.
    - Interfaces between components.
    - Logic patterns.
    - Track progress and issues.
  - State when it has been finished.

Information flow risks

University of Strathclyde Glasgow

- Inability to share thinking.
  - Lone developer – knowledge lost when developer leaves.
  - Code with no comments.
- Bad architecture design.
  - Initial prototypes might work, but final version fails.
- Bad interface design.
  - Cannot efficiently test software components.
  - Changing interface breaks software.

Information flow risks

- Bad issue tracking.
  - Not sure when a software defect has been fixed.
- No definition of completion.
  - Continue to develop software, which is not accepted.
  - Cannot invoice for completed work.

Development Approaches

Prototyping: developer led

Invest to derisk before committing to build.

- Understand hardware.
  - Build drivers to access functionality.
- Understand software framework.
- Test core algorithms.
- Performance testing.
  - Memory, CPU, disk, network use.
  - Scale to deployment – affects hardware choice.

Prototyping: user led

Implicitly committed to build.

- Understand requirements.
  - User look and feel.
  - Interactive design.
  - Deploy prototype version for user testing.

Building up

University of Strathclyde Glasgow

- Following hardware selection:
  - Interface with I/O peripherals.
  - Get inputs, write to outputs.
  - Display data on screen or send to communication bus.
  - Generic functions, build up to level needed for project.
  - Data structures defined by hardware.

Building down

- Once user interface has been agreed:
  - Get inputs and write outputs.
  - Actions driven by user interaction.
  - Data structures defined by user interface.

Filling the gap

- Functionality to satisfy customer/stakeholder.
- Scope set by development budget/cost.
- Re-use low-level functionality as needed.
  - Can cause more low-level development.

University of Strathclyde Glasgow

Development Lifecycles

Software development lifecycle

One iteration

- Define the problem – user requirements.
- Define the user interface.
- Define the high-level design.
- Define the components of the program.
- Implement the program.
- Test the program – check requirements are fulfilled.
- Deploy and operate.

Software development lifecycle

One iteration

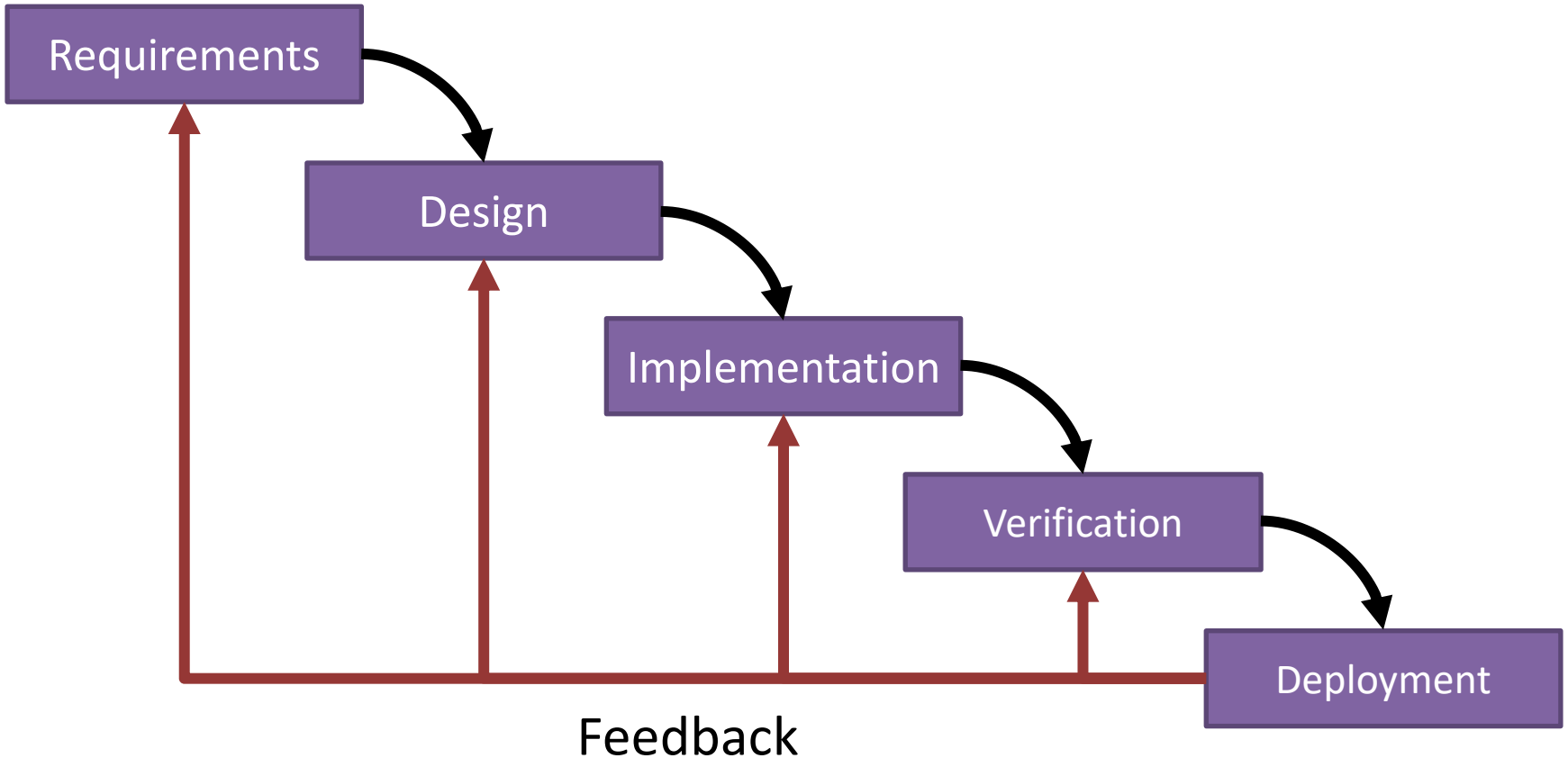
- Requirements definition.
- Software and systems design.
- Implementation and unit testing.
- Integration and system testing.
- Operation and maintenance.

Traceability

Bidirectional mapping

- Feature definition.
  - Why it is needed.
  - Who requested it.
- Feature validation.
  - How it has been tested.
- Release contents:
  - Requested features.
  - Dormant features – extensibility or exploits.
    - Safety critical systems require reverse mapping.

Lifecycles: Waterfall



W. W. Royce, “Managing the Development of Large Software Systems,” Proceedings IEEE WESCON, Los Angeles, 25-28 August 1970, pp. 1-9.

W. W. Royce, “Managing the development of large software systems: concepts and techniques”, *Proceedings of the 9th international conference on Software Engineering*. 1987.

Lifecycles: Waterfall

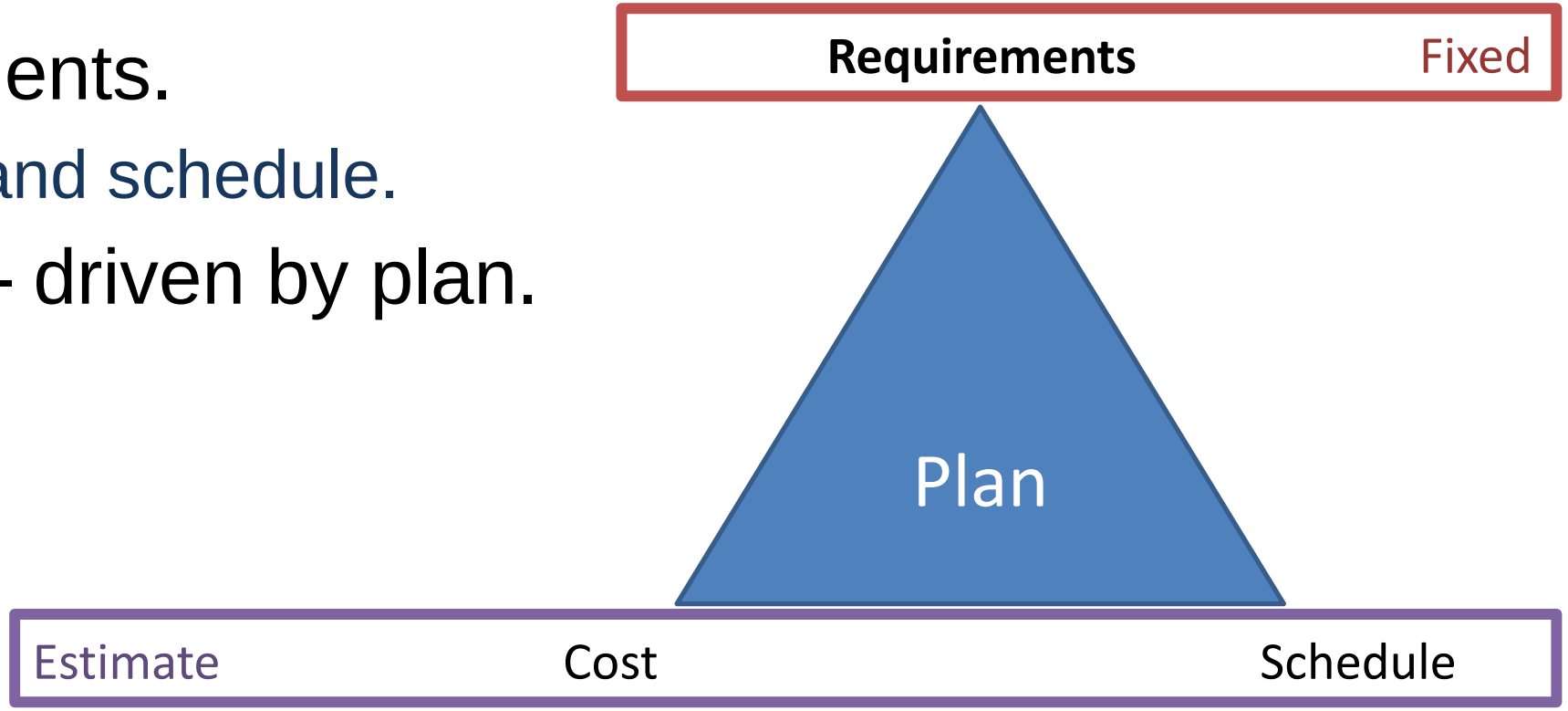
University of Strathclyde Glasgow

- Requirements are fixed.
  - Long gap between requirements and delivery.
  - Requirements may not be suitable at delivery.

- Stakeholders are unable to alter development.
  - Unhappy customers.
  - Easier to understand cost of development.

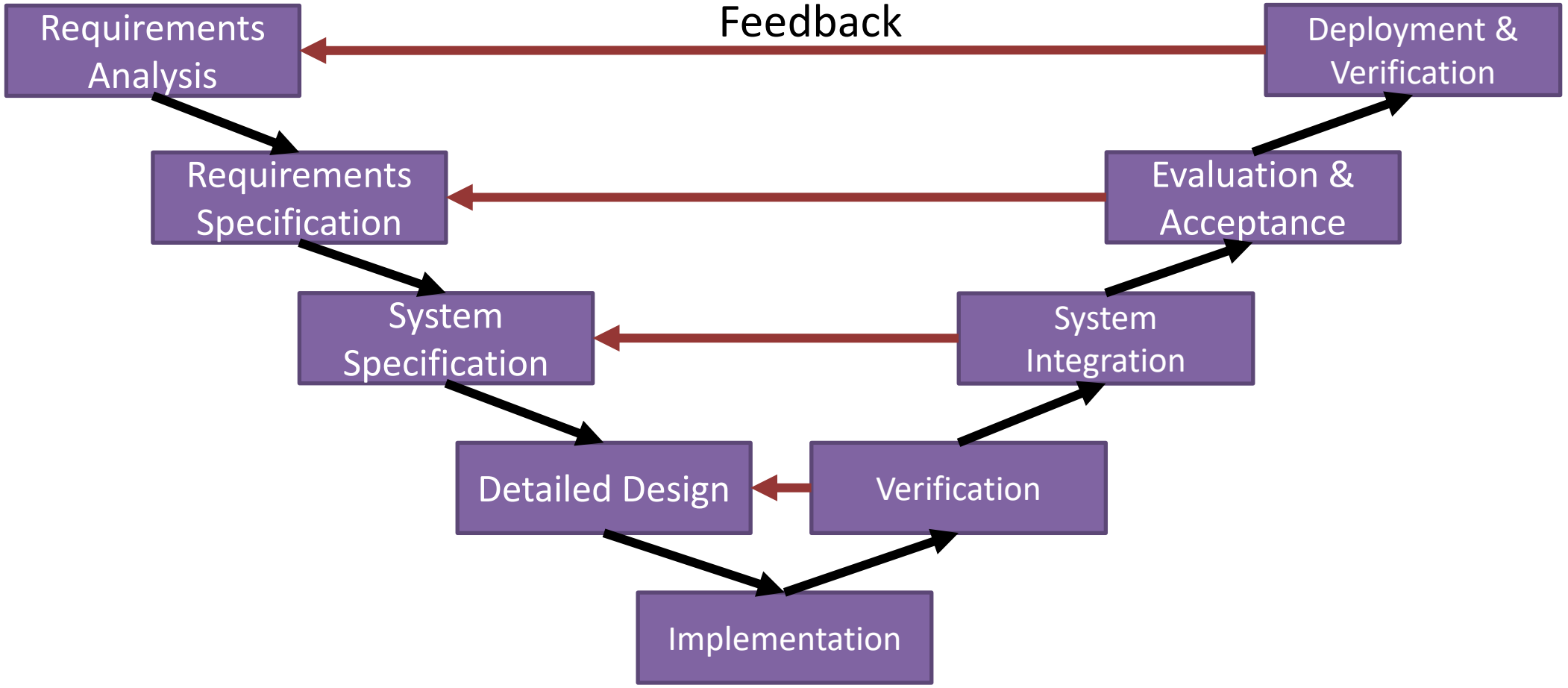
Plan Driven

- Fixed requirements.
  - Estimate cost and schedule.
- “Iron triangle” – driven by plan.



Dean Leffingwell, “Agile Software Requirements”, Addison-Wesley, 2011

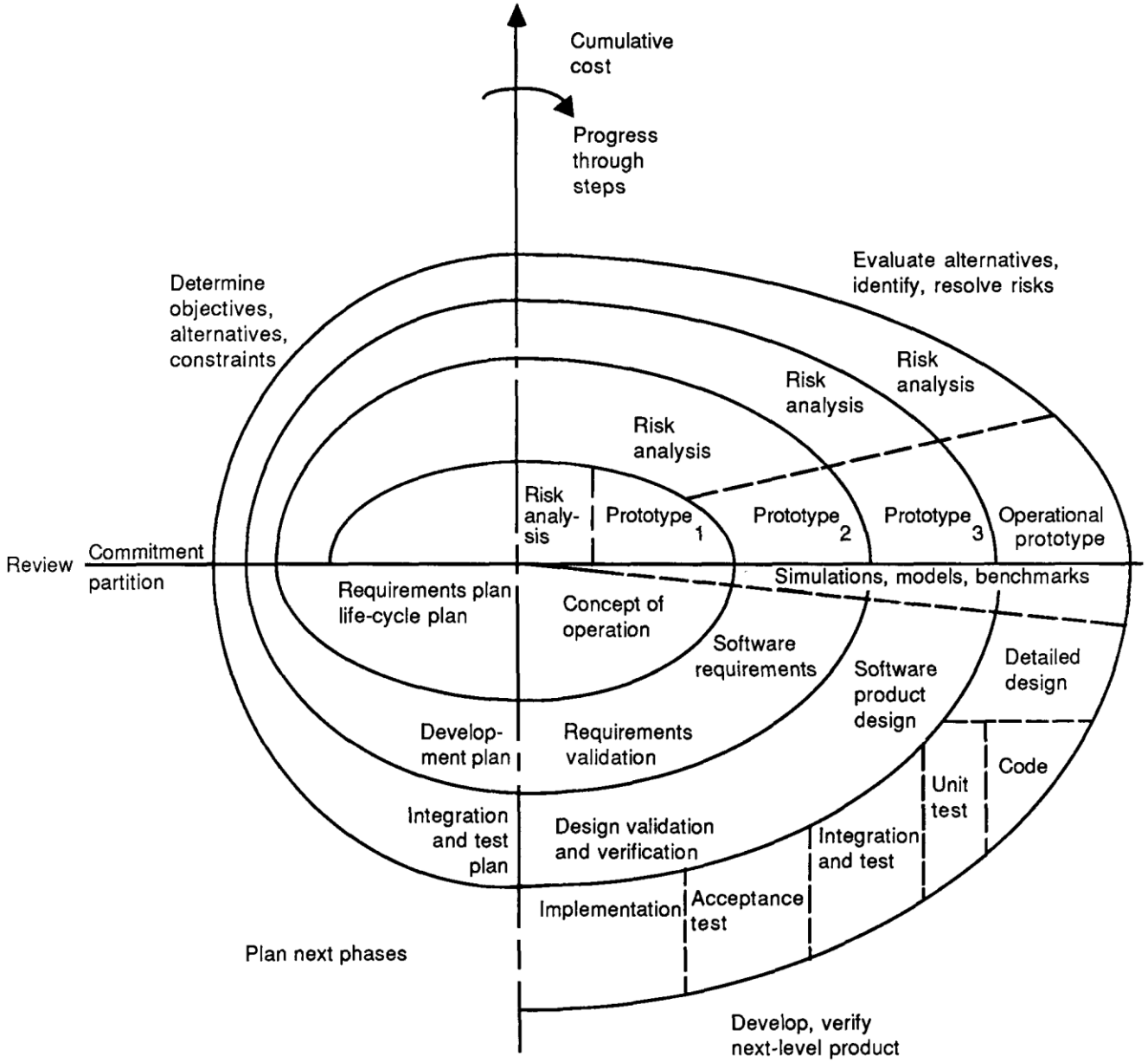
Lifecycles: V-lifecycle



Lifecycles: V-lifecycle

- Dwindling interest, due to inflexibility.
- Still some interest to constrain costs.
- Can be used for initial build.
  - Better for smaller software projects.

Lifecycles: Spiral



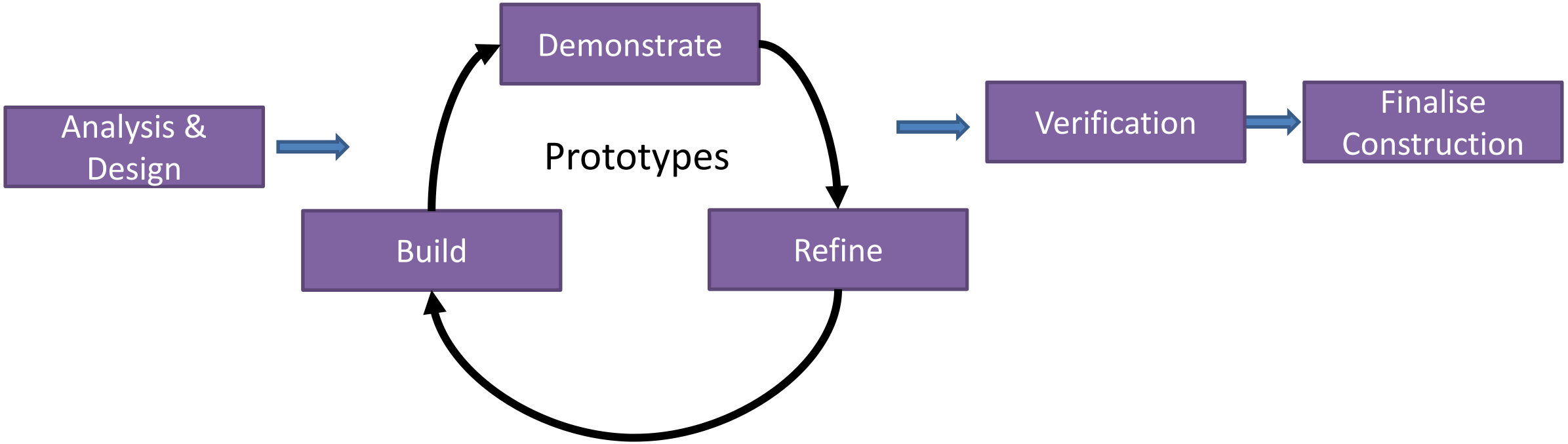
B. W. Boehm, "A spiral model of software development and enhancement", *Computer* 21.5 (1988): 61-72.

Lifecycles: Spiral

University of Strathclyde Glasgow

- Produce prototypes to understand requirements.
  - Use interaction with prototype.
- Outer layer of spiral follows waterfall steps.
  - Design, implementation, testing, acceptance.

Lifecycles: Rapid Application Development



J. Martin, "*Rapid application development",* Macmillan Publishing Co., Inc., 1991.

University of Strathclyde Glasgow

Agile Lifecycles

Agile Software Manifesto

University of Strathclyde Glasgow

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

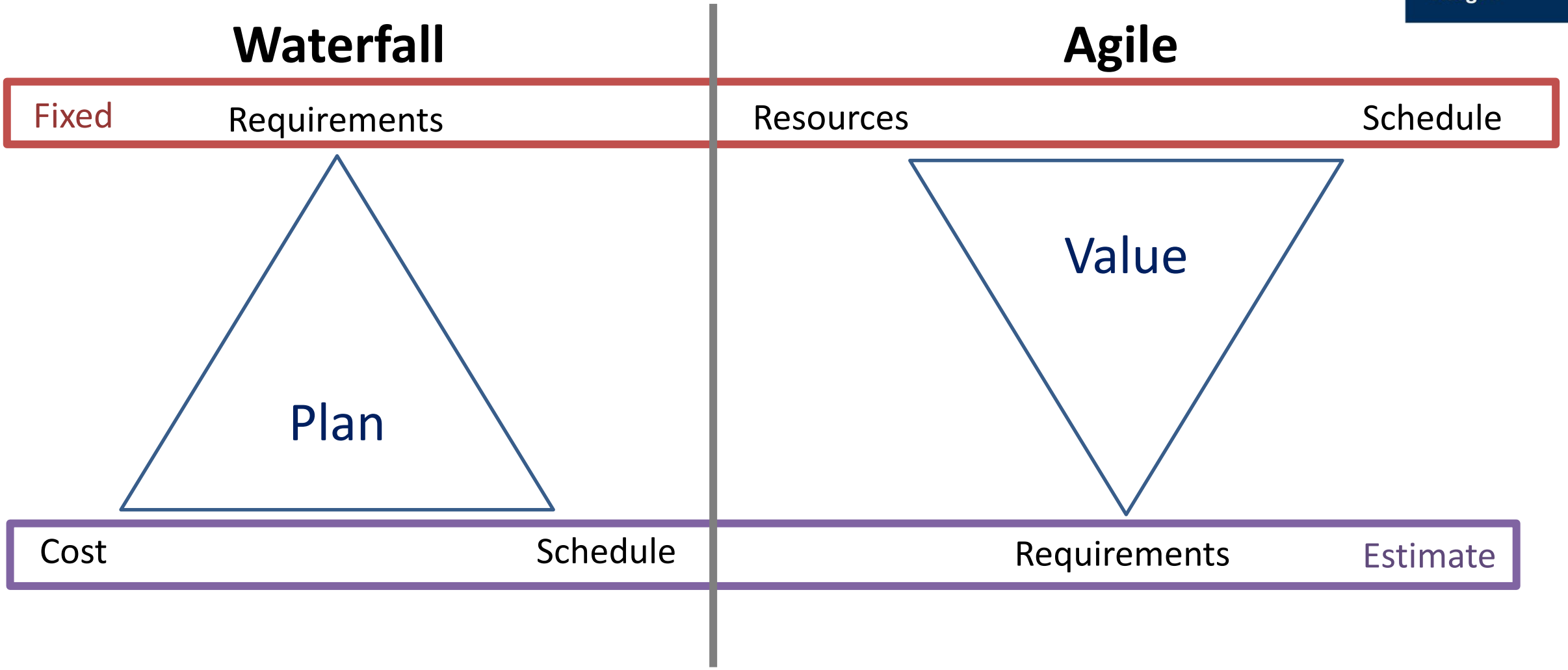
https://agilemanifesto.org/

Agile Software: Goals

University of Strathclyde Glasgow

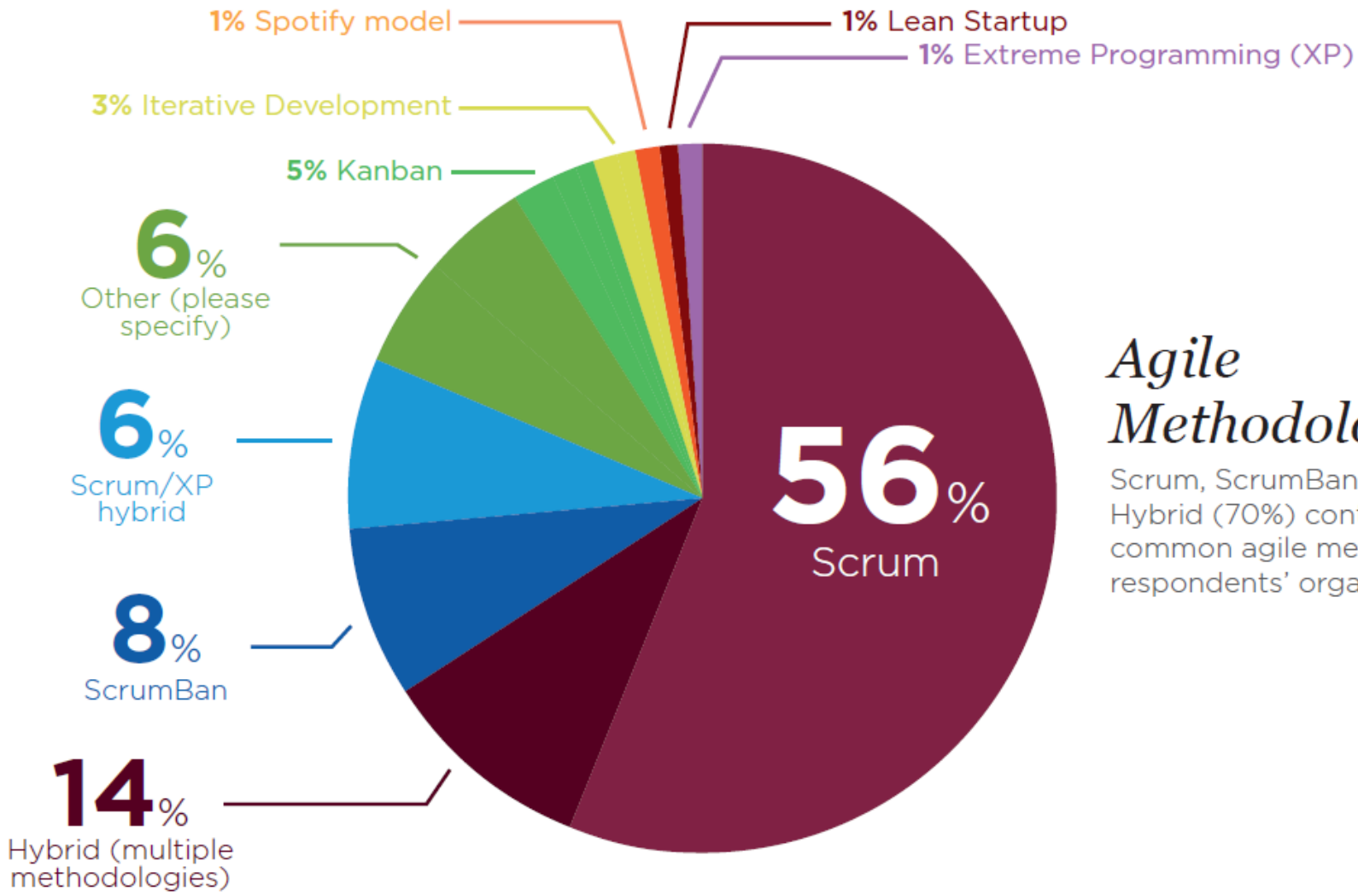
- Satisfy the customer – continuous delivery.
  - Accept late requirement changes.
- Working software iterations.
  - Deliver frequently.
- Owner and developers work together.
  - Build projects around motivated people.
  - Face-to-face meetings for high efficiency.
- Promote sustainable development.
  - Continue to produce iterations at a manageable pace.

Value Driven



Dean Leffingwell, “Agile Software Requirements”, Addison-Wesley, 2011

Agile Methodologies



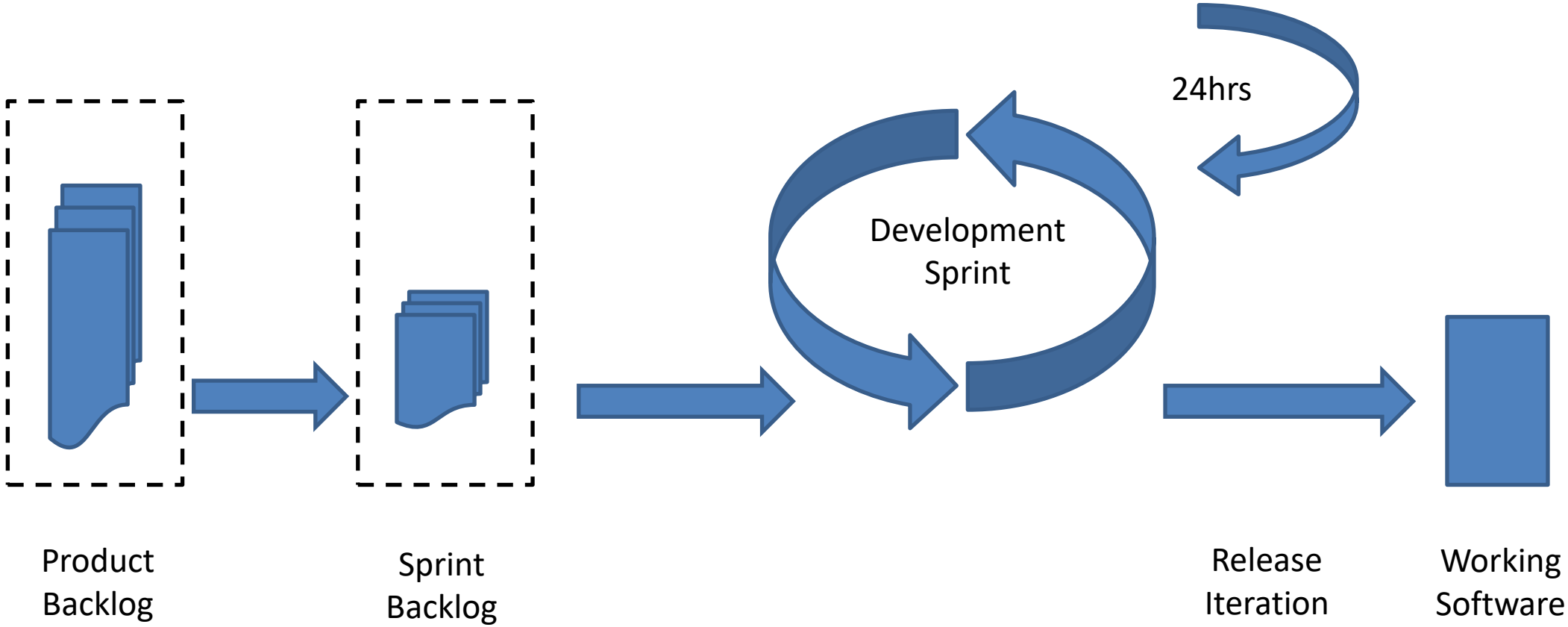
Agile Methodologies Used

Scrum, ScrumBan and Scrum/XP Hybrid (70%) continue to be the most common agile methodologies used by respondents' organizations.

12th Annual state of Agile Report, VersionOne, 2017

Agile Scrum

Passing tests indicates completion.



https://www.scrumalliance.org/

Agile Techniques

- Selected techniques.
  - Dependent on company.
  - Dependent on project.
  - Dependent on customer.

12th Annual state of Agile Report, VersionOne, 2017

University of Strathclyde Glasgow

Development Failures

Failures

University of Strathclyde Glasgow

- Software is not what client wanted.
  - Insufficient requirements capture.
- Software does not work on client system.
  - Badly designed acceptance tests.
- Software contains defects that only appear when used.
  - Faulty test suite or test system design.

Failures

- Possible to fail using any software lifecycle.
  - Lack of focus on development goals.
- Agile development may reduce chance of failure.
  - Must involve working increments.
  - Must avoid snowballing – client waits for “final” release.

Police Scotland, Failed i6 project

- Good practice in early stages.
- Contractor underestimated the complexity of project.
- Disagreement between contractor and client.
  - Loss of trust followed.
  - Disputes about the project's scope.
- Resulted in £24.65m settlement.
  - £11.09m refund.
  - £13.56m additional payment.

https://www.audit-scotland.gov.uk/news/failed-i6-project-leaves-urgent-need-to-deliver-police-it-requirements

Police Scotland, Failed i6 project

University of Strathclyde Glasgow

- Critical errors in technical coding.
- Many software defects.
  - Could not resolve them within a reasonable time limit.
  - Fixing one issue caused another to occur.
- Lack of compliance – criminal justice module.
- Error in search and audit modules.
- Limited functionality in administrative module.
  - Contractor invoiced for delivery of module.

https://www.audit-scotland.gov.uk/news/failed-i6-project-leaves-urgent-need-to-deliver-police-it-requirements

Conclusions

University of Strathclyde Glasgow

- Software engineering is vital for success.
  - Complexity of system of requirements and tests scales.
- Mostly use Agile lifecycles.
  - Many versions of “Agile” lifecycle exist.
  - Scrum is the most popular.
- Software engineering failures can be very costly.
  - Need to fail fast, within prototyping.
  - Deliver working increments.

University of Strathclyde Glasgow