# CS3213: Foundations of Software Engineering

## Software Process Models, Plan-driven and Agile Software Engineering

## Software Development

The software to meet the specification **must be produced**

## Software Evolution

the software **must evolve** to meet changing customer needs

## Software Validation

the software **must be validated** to ensure that it does what the customer wants

## Software Specification

The functionality of the software and constraints on its operation **must be defined**

# Central Software Engineering Activities

**Software Specification**

The functionality of the software and constraints on its operation **must be defined**

**Software Development**

The software to meet the specification **must be produced**

**Software Validation**

the software **must be validated** to ensure that it does what the customer wants

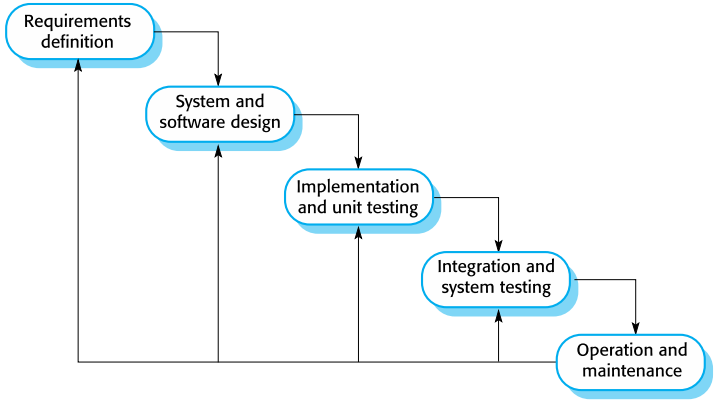
**Software Evolution**

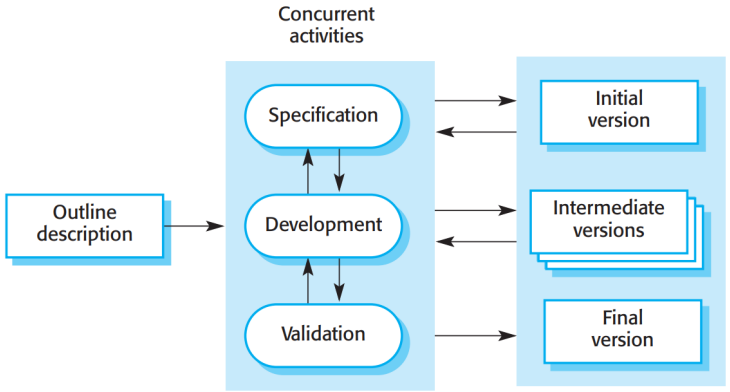
the software **must evolve** to meet changing customer needs

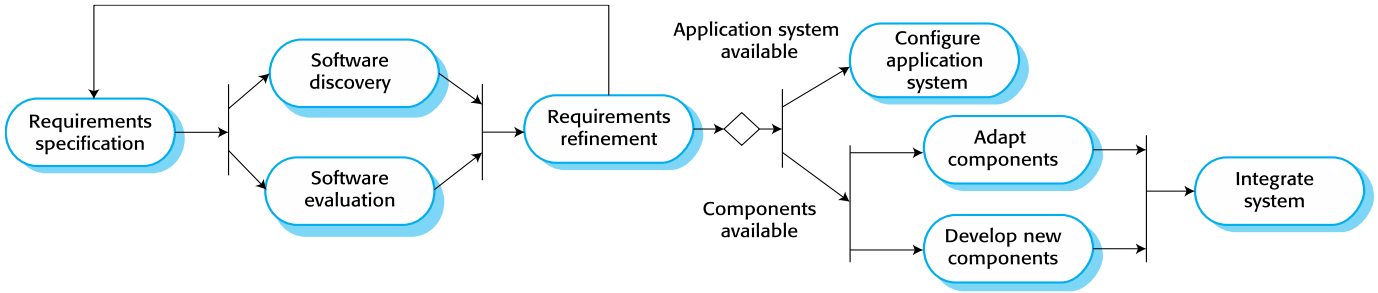
# Software Processes

- ***Software process***: a set of related activities that leads to the production of a software system
- ***Software process model***: simplified representation of a software process (e.g., sequence of activities)
  - Illustrate different approaches to developing software
- No silver bullet!

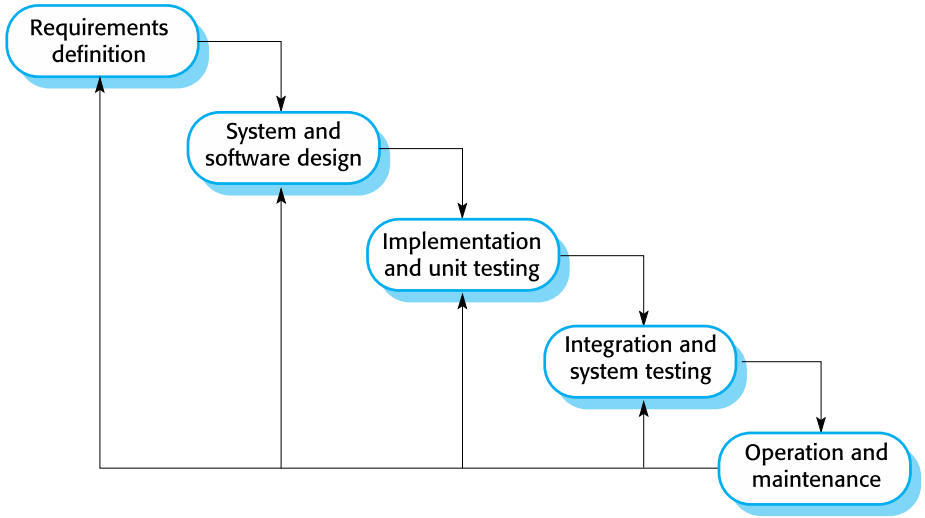
# Three Important Software Process Models



# Waterfall Model



# Waterfall Model

MANAGING THE DEVELOPMENT OF LARGE SOFTW

*Dr. Winston W. Royce*

## INTRODUCTION

I am going to describe my personal views about managing large softw
various assignments during the past nine years, mostly concerned with the dev
for spacecraft mission planning, commanding and post-flight analysis. In these
different degrees of success with respect to arriving at an operational state, on
become prejudiced by my experiences and I am going to relate some of these

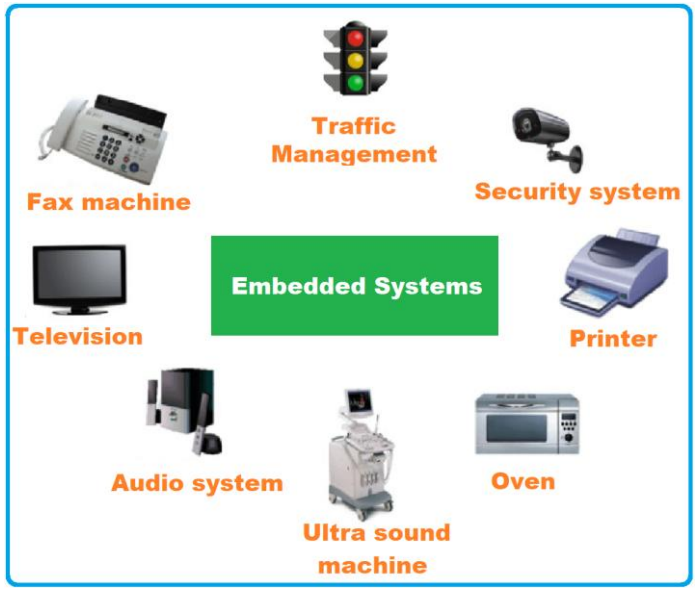
## COMPUTER PROGRAM DEVELOPMENT FUNCTIONS

There are two essential steps common to all computer program developments, regardless of size or complexity. There is first an analysis step, followed second by a coding step as depicted in Figure 1. This sort of very simple implementation concept is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it — as is typically done with computer programs for internal use. It is also the kind of development effort for which most customers are happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product. An implementation plan to manufacture larger software systems, and keyed only to these steps, however, is doomed to failure. Many additional development steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development costs. Customer personnel typically would rather not pay for them, and development personnel would rather not implement them. The prime function of management is to sell these concepts to both groups and then enforce compliance on the part of development personnel.

*"[…] the implementation [of the waterfall model] described above is **risky and invites failure**. […] The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed."*

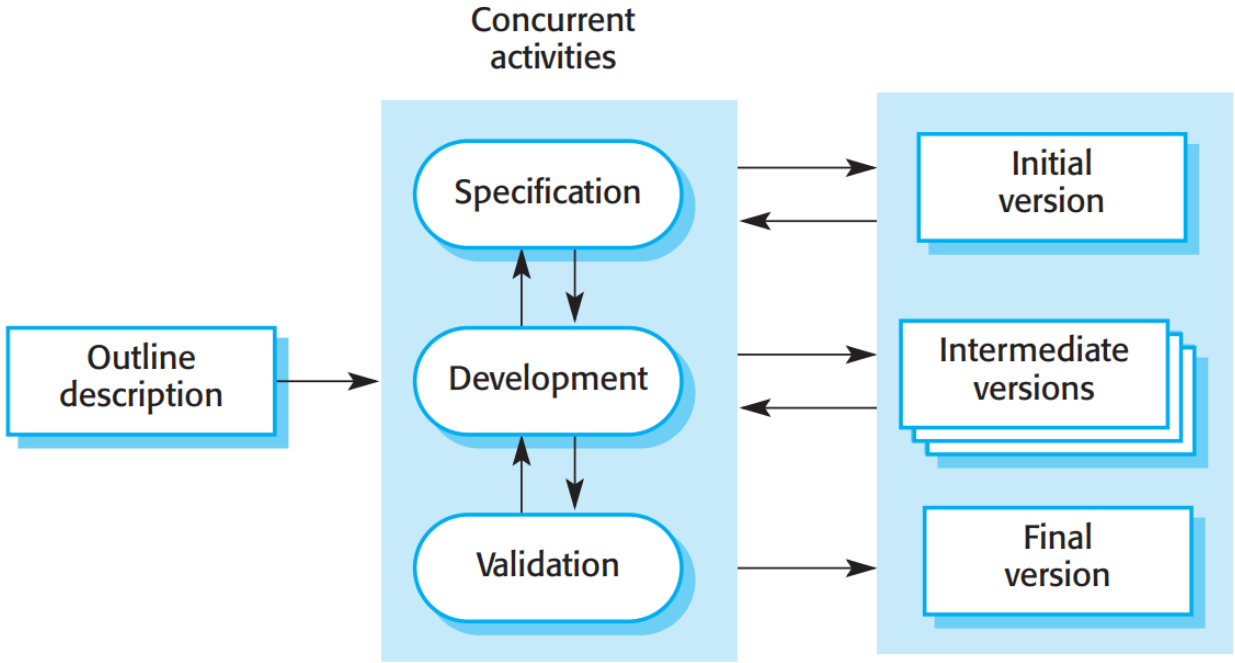
https://dl.acm.org/doi/pdf/10.5555/41765.41801

# Use Cases




https://www.theengineeringprojects.com/2021/06/real-time-embedded-systems-definition-types-examples-and-applications.html

# Incremental Development



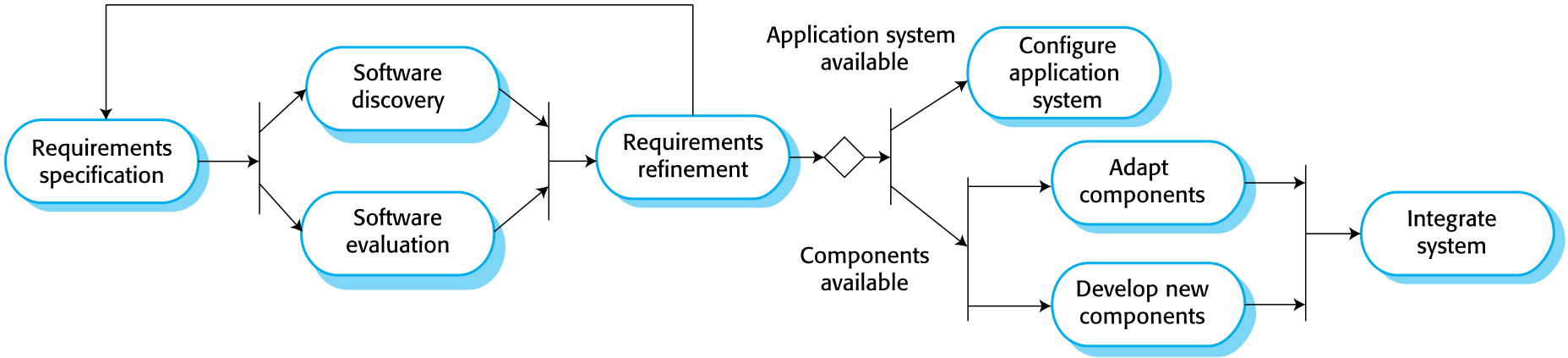
# Incremental Development

- Pros
  - Cost of accommodating change and responding to changing customer requirements is lower
  - More rapid delivery of software
- Cons
  - Process is less measurable (for managers)
  - Degrading system structure (refactoring required)

# Integration and Configuration

- Observation: many existing (open-source) systems, libraries, frameworks, …
- Idea: focus the process on re-using them, rather than developing software from the scratch
- This approach is “reuse based development”

# Reuse-based Development

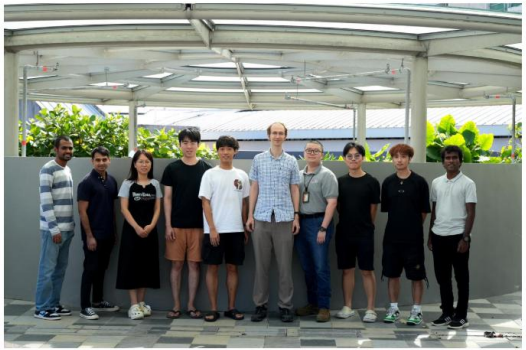


# NUS Test Page

People Projects Publications Found Bugs Openings Seminars

Trustworthy Engineering of Software Technologies Lab

The *Trustworthy Engineering of Software Technologies (TEST) Lab*, part of the PL/SE group at the National University of Singapore (School of Computing), led by Manuel Rigger, is working on practical and conceptual software solutions. We aim to have a real-world impact both by creating practical tools as well as by designing principled, fundamental techniques.



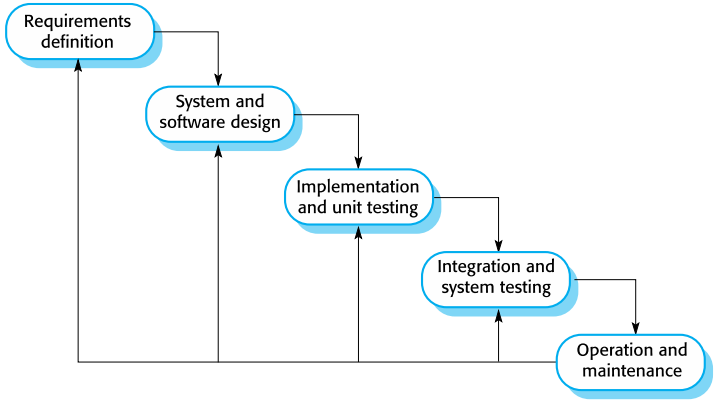
Core Areas:

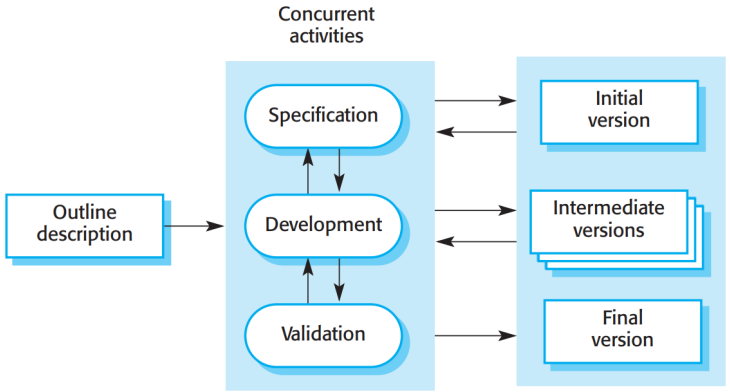
- Software Engineering
- Systems
- Programming Languages

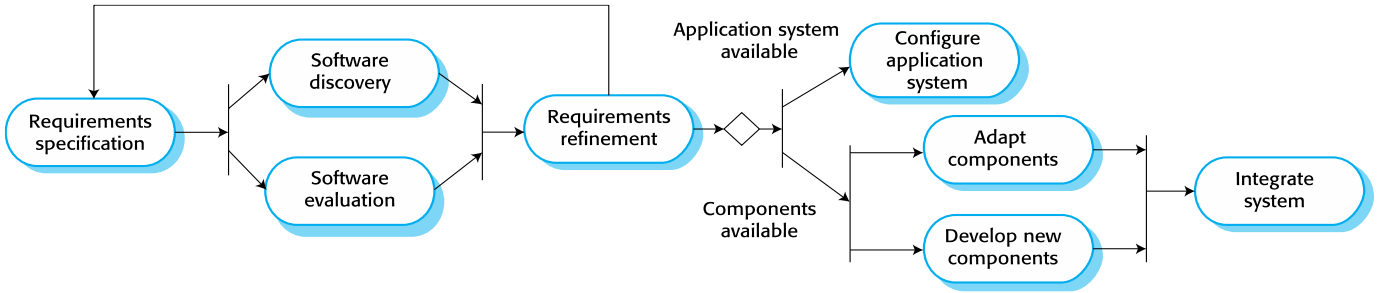
# Reuse-based Development

- Pros
  - Reduces the amount of software to be developed, which has advantage in terms of cost and managing risk
  - Faster delivery of software
- Cons
  - Requirements compromises
  - Control over system evolution is lost or might be difficult

# Three Important Software Process Models



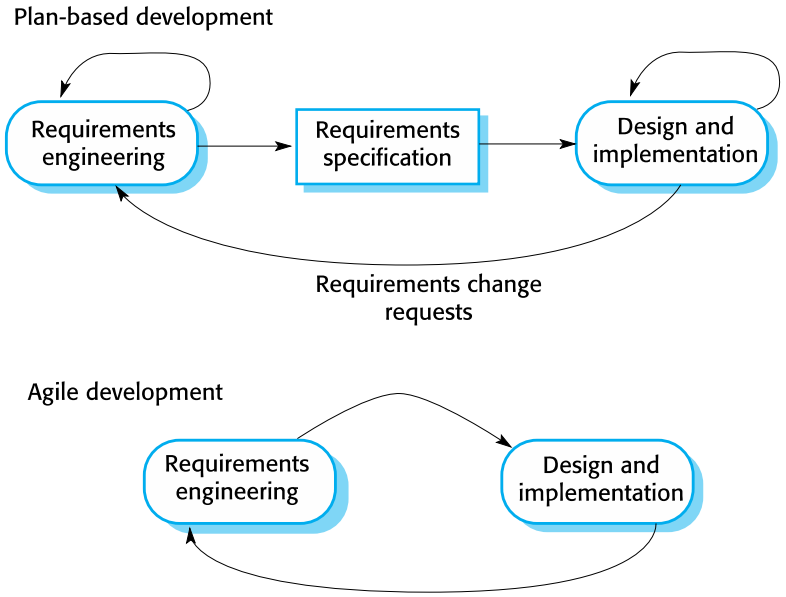



# Summary and Key Points

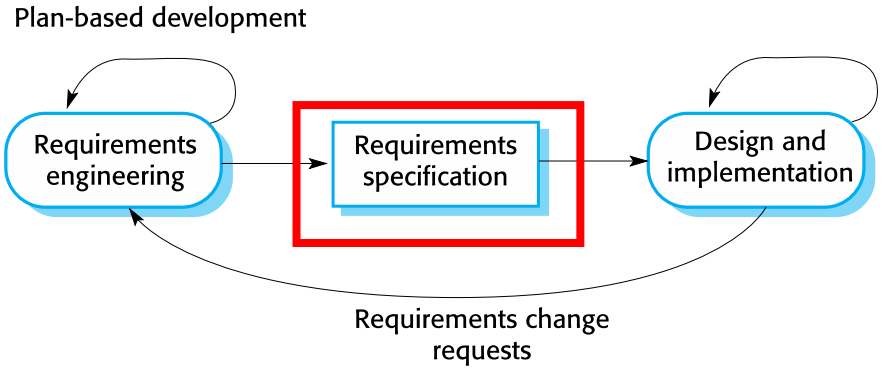
- *Software process models* specify how essential software engineering activities are executed
- Three important and common models with distinct advantages and disadvantages
  - Waterfall Model
  - Incremental Development
  - Integration and Configuration

# Plan-driven vs. Agile




Two general approaches to develop software: plan-driven and agile

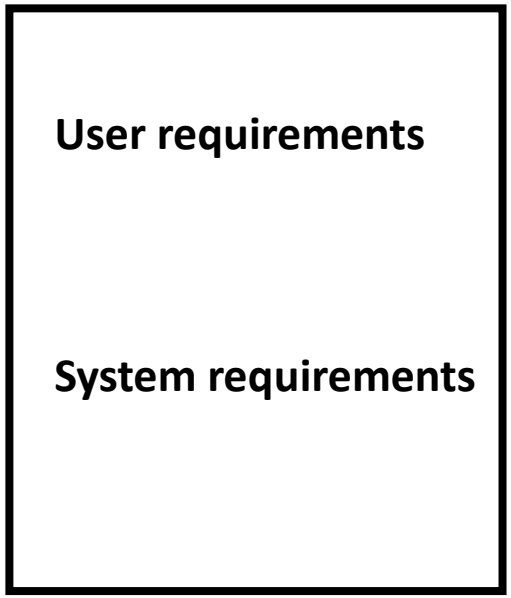
# Plan-driven vs. Agile




*“Plan your work, work your plan.”*

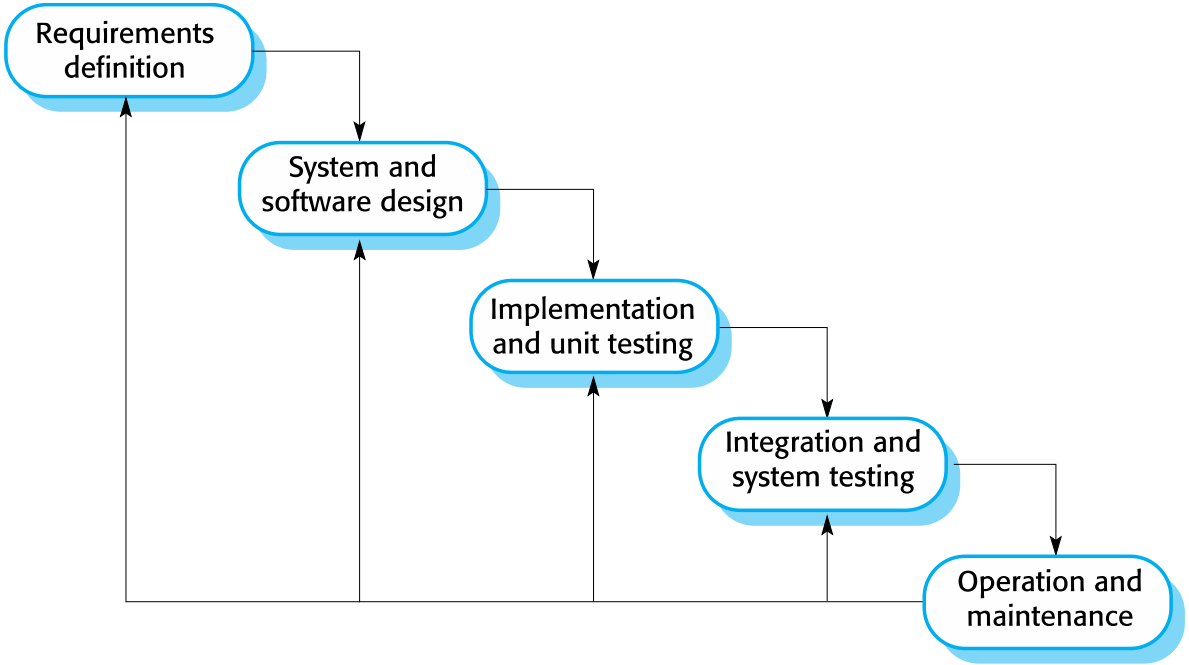
# Software Requirements Document

Software Requirements Specification (SRS)

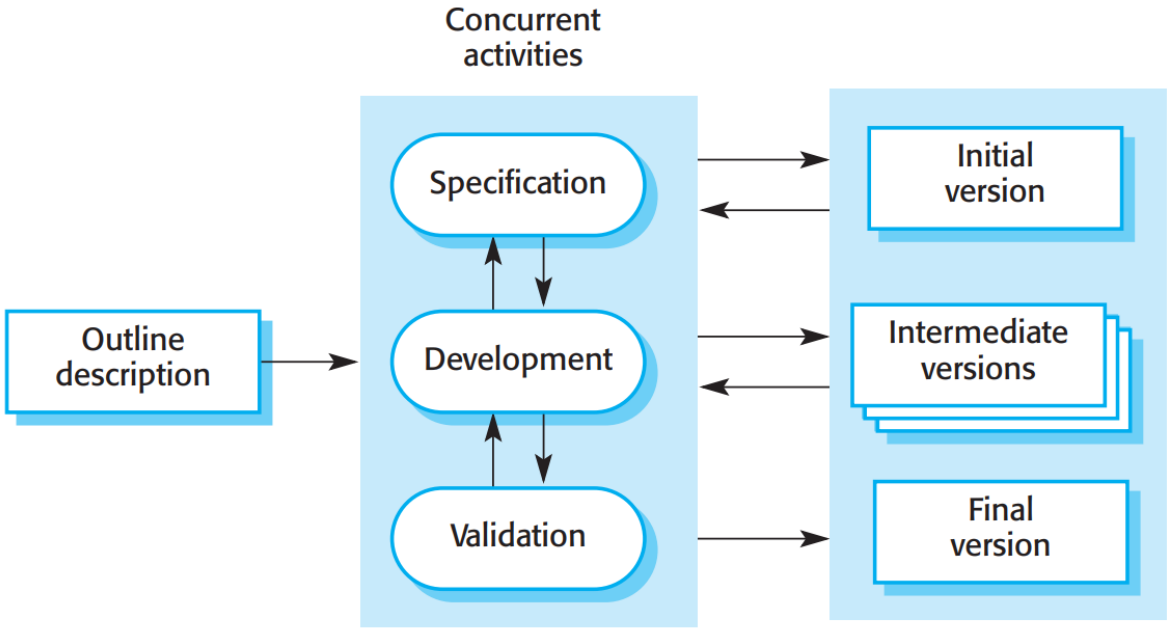


| Chapter | Description |
|---|---|
| Preface | This defines the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version. |
| Introduction | This describes the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software. |
| Glossary | This defines the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader. |
| User requirements definition | Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified. |
| System architecture | This chapter presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted. |
| System requirements specification | This describes the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined. |
| System models | This chapter includes graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models. |
| System evolution | This describes the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system. |
| Appendices | These provide detailed, specific information that is related to the application being developed—for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data. |
| Index | Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on. |

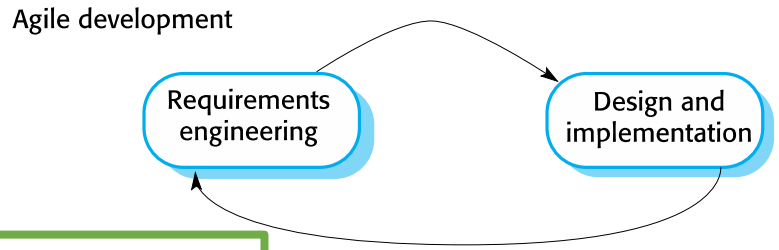
# Waterfall Model



# Incremental Development



# Plan-driven vs. Agile




*“Inspect and adapt.”*

*“Responding to change over following a plan.”*

# Agile Development (1/2)

- Both a **mindset** and **concrete management and engineering frameworks**
- Focus on **delivering value** in increments and/or avoiding waste
- **Feedback loops** and customer involvement
- **Adaptive** to changing requirements

# Agile Development (2/2)

- Lightweight processes
- Empowering teams and people
- Inspired by lean manufacturing principles
- A lot of buzzwords and certifications

# Agile Development: Historic Development

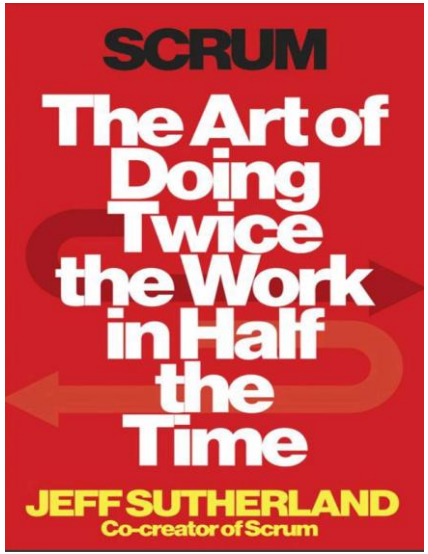
- 1980s and early 1990s: there was a widespread view that the best way to achieve better software was through…
  - **careful project planning**
  - **formalized quality assurance**
  - use of **analysis and design methods** supported by software tools
  - **controlled and rigorous software development processes**

# Agile Development: Historic Development

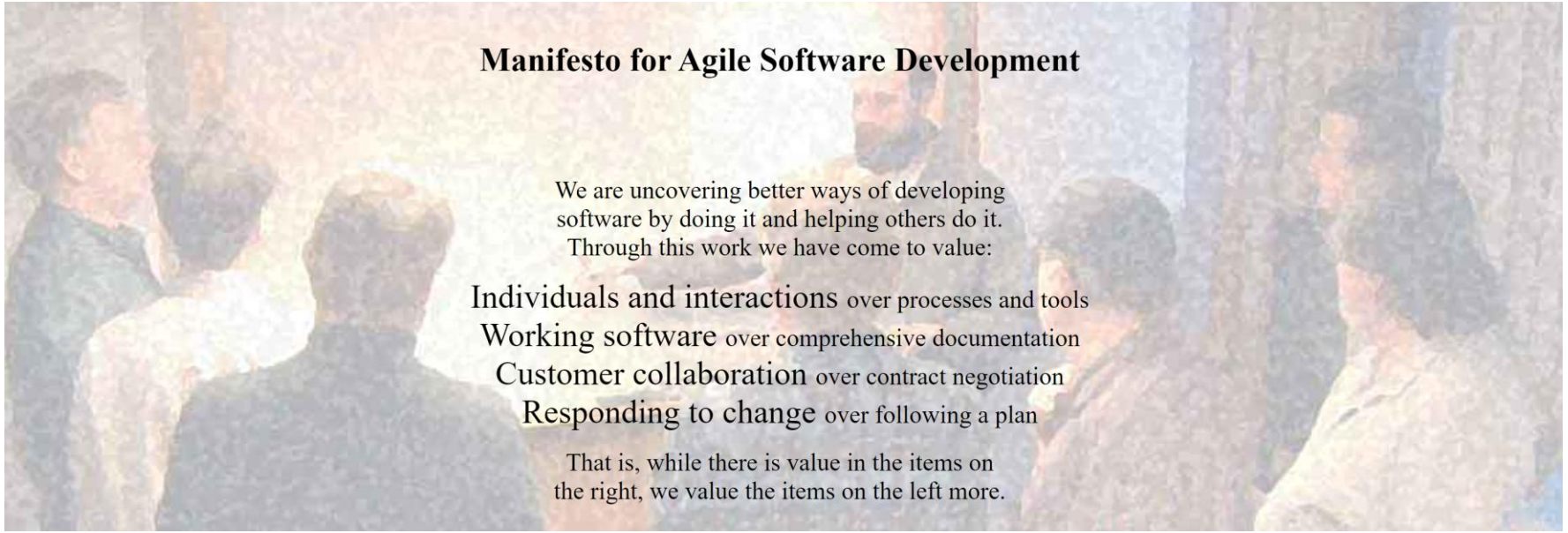
By CIA - Image 2-1, from Archangel: CIA's Supersonic A-12 Reconnaissance Aircraft, Public Domain, https://commons.wikimedia.org/w/index.php?curid=9927307

# Agile Development: Historic Development

*Up to that point—and even as late as 2005—most software development projects were created using the Waterfall method, where a project was completed in distinct stages and moved step by step toward ultimate release to consumers or software users. The process* ***was slow, unpredictable, and often never resulted in a product that people wanted or would pay to buy****.* ***Delays*** *of months or even years were endemic to the process. The early step-by-step plans, laid out in comforting detail in Gantt charts, reassured management that we were in control of the development process—but almost without fail, we would fall quickly behind schedule and disastrously over budget.*



# Agile Manifesto (2001)

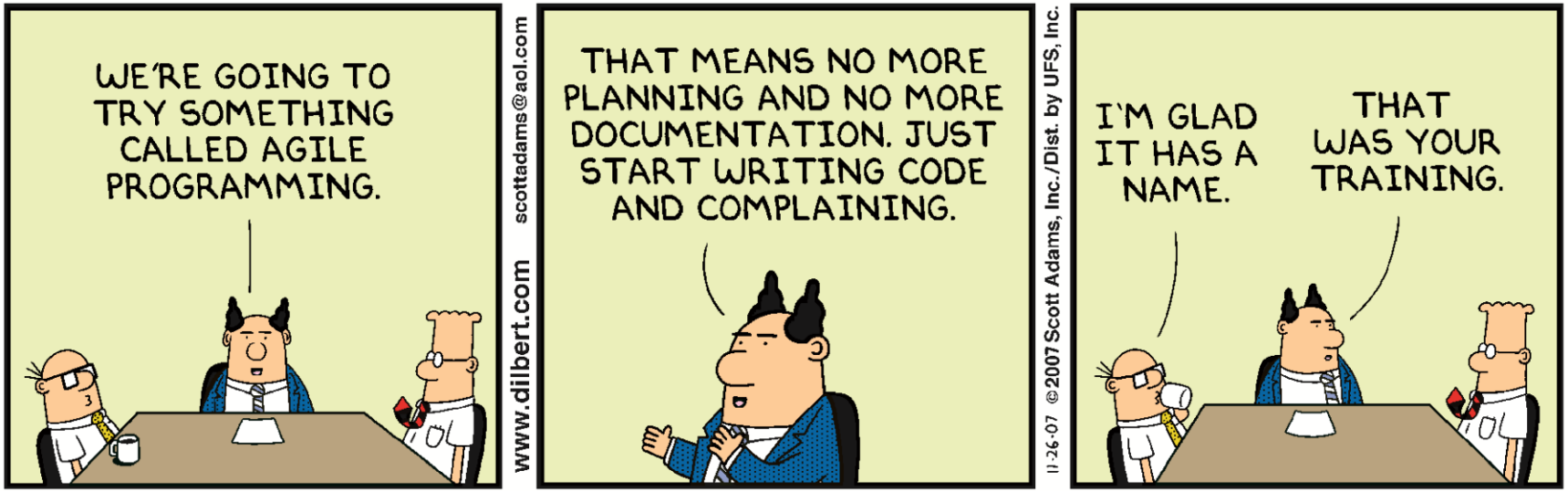



https://agilemanifesto.org/

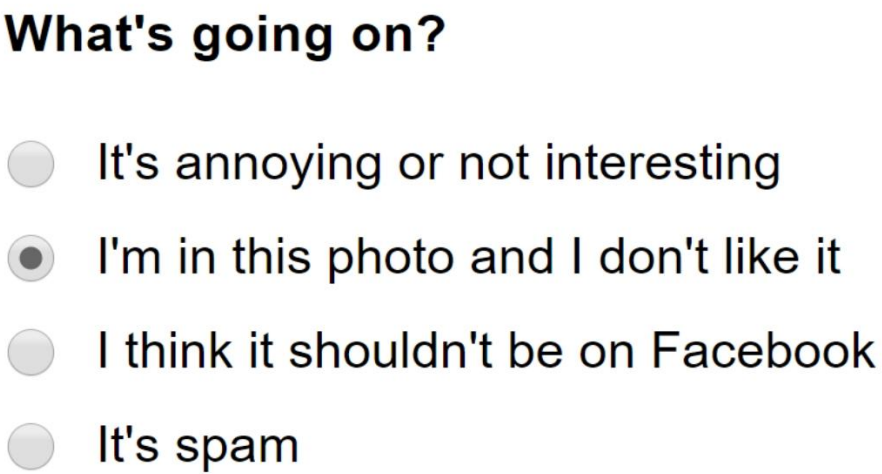
# Agile Manifesto


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

# Agile Manifesto

# Agile Manifesto



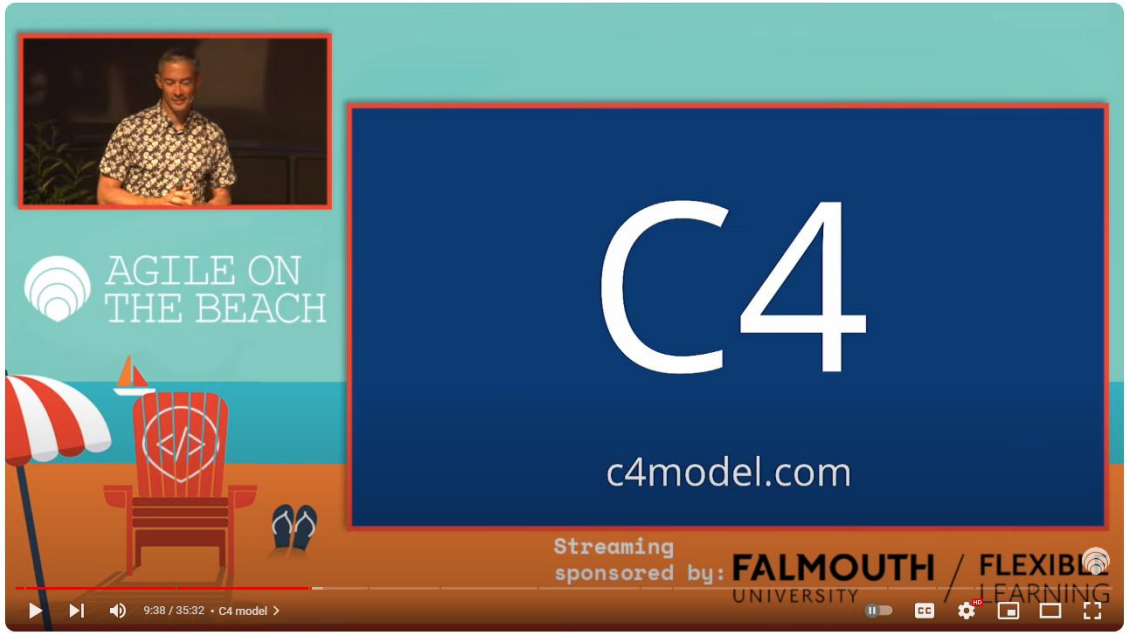

https://knowyourmeme.com/memes/im-in-this-photo-and-i-dont-like-it


# Agile Manifesto

- **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."*


# C4




Visualising software architecture with the C4 model - Simon Brown, Agile on the Beach 2019

Agile on the Beach
3.71K subscribers
Subscribe
5.1K
Share
Download
Clip
Save

Agile software development can also involve architectural planning and modeling

https://www.youtube.com/watch?v=x2-rSnhpw0g

# Agile Principles in the Agile Manifesto (1/3)


- Our highest priority is to **satisfy the customer** through **early and continuous delivery of valuable software**.
- **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- **Business people and developers must work together daily** throughout the project.

# Agile Principles in the Agile Manifesto (2/3)

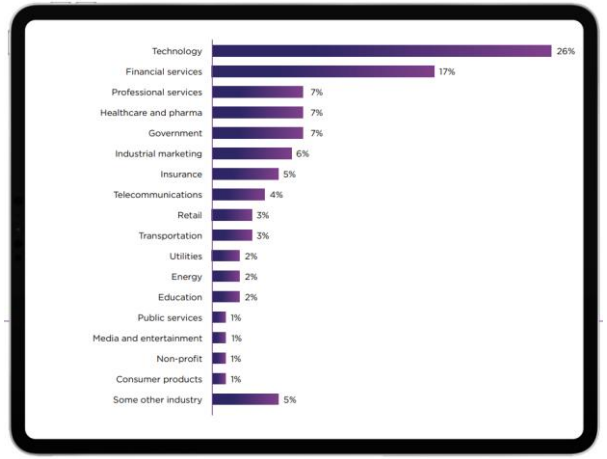

- **Build projects around motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.
- **Working software** is the primary measure of progress.
- Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

# Agile Principles in the Agile Manifesto (3/3)


- Continuous attention to **technical excellence and good design** enhances agility.
- **Simplicity**—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams**.
- At regular intervals, **the team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly.

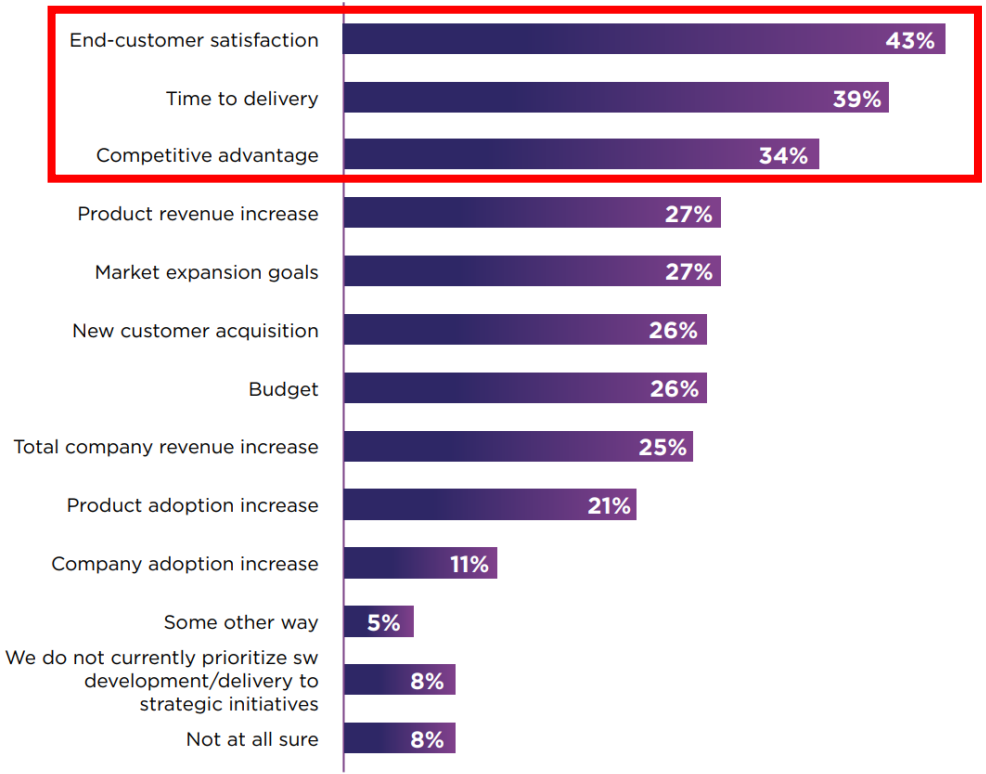
# Agile: Reasons


- Yearly survey studying the adoption of agile methods
- Survey conducted in 2023 based on 788 survey respondents

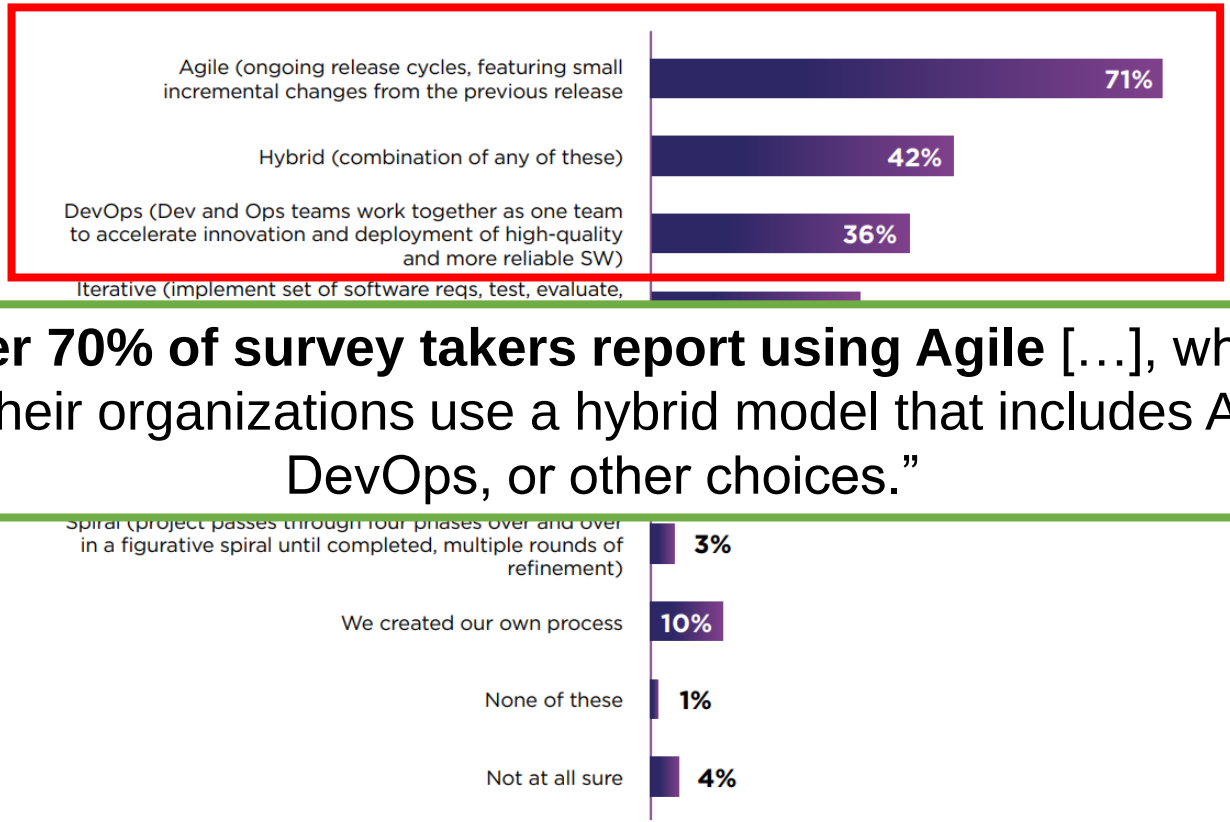



https://info.digital.ai/rs/981-LQX-968/images/RE-SA-17th-Annual-State-Of-Agile-Report.pdf

# Agile: Organizations' SE Priorities



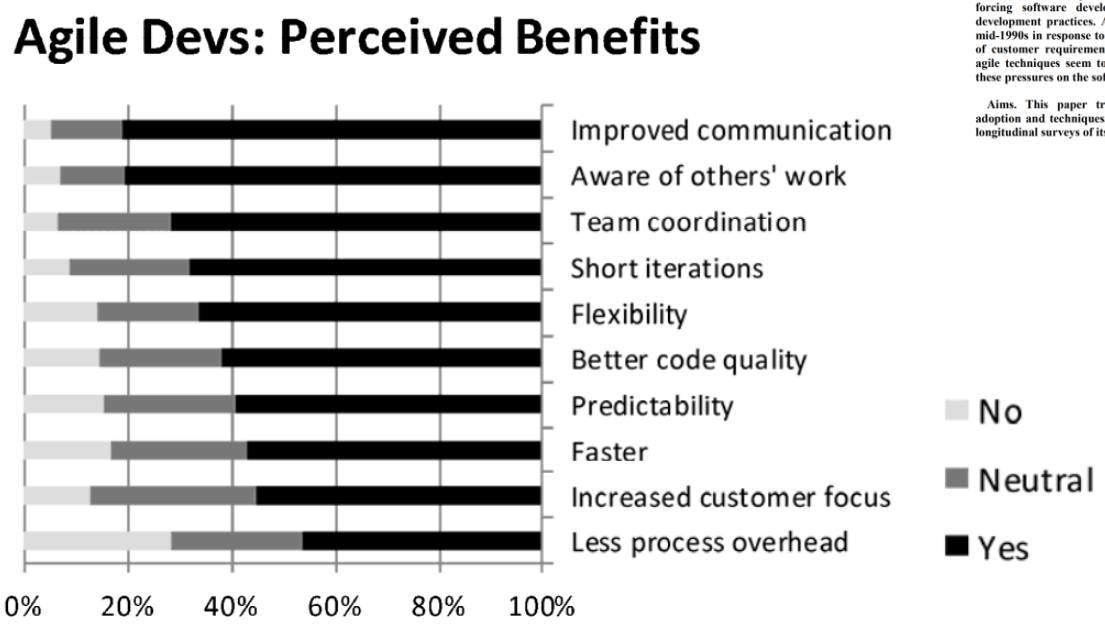
# Agile: Use




"Just **over 70% of survey takers report using Agile** […], while 42% said their organizations use a hybrid model that includes Agile, DevOps, or other choices."

# Agile: Perceived Advantages

# Agile: Microsoft




**Have Agile Techniques been the Silver Bullet for Software Development at Microsoft?**


Brendan Murphy
Microsoft Research
Cambridge, UK
bmurphy@microsoft.com

Christian Bird
Microsoft Research
Redmond, USA
cbird@microsoft.com

Thomas Zimmermann
Microsoft Research
Redmond, USA
tzimmer@microsoft.com

Laurie Williams
NCSU
Raleigh, USA
williams@csc.ncsu.edu

Nachiappan Nagappan
Microsoft Research
Redmond, USA
nachin@microsoft.com

Andrew Begel
Microsoft Research
Redmond, USA
andrew.begel@microsoft.com



***Abstract*—Background. The pressure to release high-quality, valuable software products at an increasingly faster rate is forcing software development organizations to adapt their development practices. Agile techniques began emerging in the mid-1990s in response to this pressure and to increased volatility of customer requirements and technical change. Theoretically, agile techniques seem to be the silver bullet for responding to these pressures on the software industry.**

**Aims. This paper tracks the changing attitudes to agile adoption and techniques, within Microsoft, in one of the largest longitudinal surveys of its kind (2006-2012).**


dology was used to develop these large software products. In reality, many large software companies did not religiously follow any specific development methodology and adapted methods and tools to suit the products they were producing.

Over time, consumers of software and software-intensive products increasingly welcomed more frequent software releases. Simultaneously, software began to be distributed electronically, and software-as-a-service (SaaS) increased in popularity. Traditional methodologies were viewed as too slow, not customer focused, not adaptable and too bureaucratic to handle the new software reality. In response, agile methods emerged in

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Agile20Trends20ESEM20Master.pdf

# Summary and Key Points

- At a high level, you can choose between a plan-driven and agile approach to software engineering
  - Plan-driven models focus on producing a high-quality SRS, based on which the system is developed
  - Agile approaches account for changing requirements by focusing on producing value (e.g., working software) quickly
- The Agile Manifesto specifies the key agile principles

# Main Sources and References