Introduction to Software Processes

How do we build software systems?

$$\begin{tikzcd}\boxed{\text{Specification}} \arrow[r, dashed] & \boxed{\text{Implementation}}\end{tikzcd}$$

The **specification** may be in many forms:

- an idea in our head,
- a set of requirements written in English,
- a formal specification.

The **implementation** will be a software system.

The arrow represents the approach we use, often called a **Software Development Process.**

Is Software Process important?

Ever since we started building more complicated software systems (c. 1960s), there has been an interest in **software development processes**.

The "holy grail" in this area is a process that transforms a specification into an implementation.

Formal Methods solve this problem for simple systems, but this doesn’t address real systems building issues.

Software process remains a fluid area, largely because we aren’t (ever) satisfied with its success rate.

Software Failure

It is a popular complaint (true or not) that software projects typically:

- run overtime
- run over budget
- fail to meet their requirements,

if indeed they manage to produce a working product at all.

Some software projects fail very dramatically, and there are many examples. We'll look at just one briefly.

AT&T 1990 Network Crash

AT&T’s long distance network crashed and remained down for nine hours in January, 1990.

A failure in one switch caused a propagation of error to all the other switches.

75,000,000 phone calls failed and 200,000 airline reservations were lost.

The failure was traced back to the C coding error on the next slide.

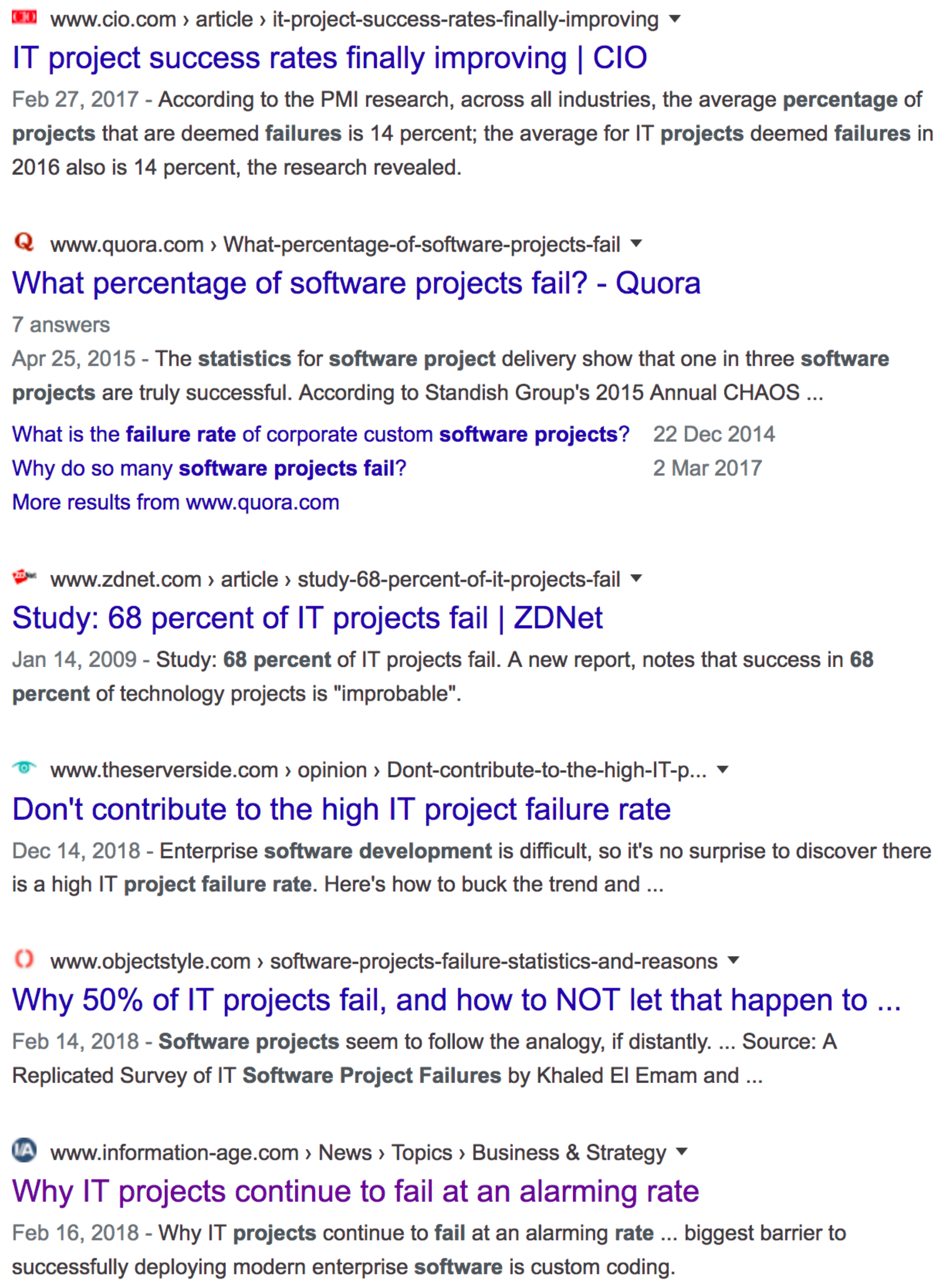
Source of the AT&T Network Crash

```
while (expression){
  switch expression{
    ...
    case (value):{
      if (logical) {
        sequence of statements
        break
      }
      else {
        other statements
      }
      statements after if...else statement
    }
  }
  statements after switch statement
}
statements after while statement
```

What is the effect of this break statement?

Note: The switch in question was the AT&T 4ESS model. 100 million lines of C code were reportedly written in the development of its successor, the 5E55

Let’s Google “software failure”



www.cio.com › article › it-project-success-rates-finally-improving
IT project success rates finally improving | CIO
Feb 27, 2017 - According to the PMI research, across all industries, the average **percentage** of **projects** that are deemed **failures** is 14 percent; the average for IT **projects** deemed **failures** in 2016 also is 14 percent, the research revealed.

www.quora.com › What-percentage-of-software-projects-fail
What percentage of software projects fail? - Quora
7 answers
Apr 25, 2015 - The **statistics** for **software project** delivery show that one in three **software projects** are truly successful. According to Standish Group's 2015 Annual CHAOS ...
What is the **failure rate** of corporate custom **software projects**? 22 Dec 2014
Why do so many **software projects fail**? 2 Mar 2017
More results from www.quora.com

www.zdnet.com › article › study-68-percent-of-it-projects-fail
Study: 68 percent of IT projects fail | ZDNet
Jan 14, 2009 - Study: **68 percent** of IT projects fail. A new report, notes that success in **68 percent** of technology projects is "improbable".

www.theserverside.com › opinion › Dont-contribute-to-the-high-IT-p...
Don't contribute to the high IT project failure rate
Dec 14, 2018 - Enterprise **software development** is difficult, so it's no surprise to discover there is a high IT **project failure rate**. Here's how to buck the trend and ...

www.objectstyle.com › software-projects-failure-statistics-and-reasons
Why 50% of IT projects fail, and how to NOT let that happen to ...
Feb 14, 2018 - **Software projects** seem to follow the analogy, if distantly. ... Source: A Replicated Survey of IT **Software Project Failures** by Khaled El Emam and ...

www.information-age.com › News › Topics › Business & Strategy
Why IT projects continue to fail at an alarming rate
Feb 16, 2018 - Why IT **projects** continue to **fail** at an alarming **rate** ... biggest barrier to successfully deploying modern enterprise **software** is custom coding.

14% failure

19% failure

68% failure

“70% of the companies polled reported having at least one project failure in the last year”

50% failure

“UK Wasting £37 Billion a Year on Failed Agile IT Projects”

What is a Software Process?

In general, we can view a software development process as comprising three main components:

A **set of models** to be produced.

A **notation** for describing these models.

A **process** to be followed.

We'll see that the Waterfall process involves creating analysis and design models, while Agile processes involve no modelling at all.

We now explore *process* in more detail.

Aside: Formal Methods

Through the 80s and early 90s there was a great interest in **Formal Methods.**

The view was that software development could never produce reliable software components.

Thus approaches based on formal specification and proof were developed:

- formal specification of requirements
- formal proof that the software was a correct implementation of the specification

What happened to Formal Methods?

The software community proved the formalists wrong: reliable software could be developed without use of formal specification/proof.

The venerable formalist, Tony Hoare, conceded this in 1996:

> "... the problem of program correctness has turned out to be far less serious than predicted... the techniques employed to achieve reliability are little different from those which have proved effective in all other branches of modern engineering..."

Note: C. A. R. Hoare, "How did software get so reliable without proof?", Invited Lecture, International Symposium of Formal Methods Europe, 1996

Roadmap of this Section

We start by looking at two “traditional” methods, both of which have considerable limitations:

Code and Fix

The Waterfall process

We then consider the key aspects of all more recent methods:

Incremental

Iterative

Finally we examine the contemporary approach:

Agile Processes

(e.g. Extreme Programming, Scrum, Kanban etc.)

"Code and Fix" (aka hacking)

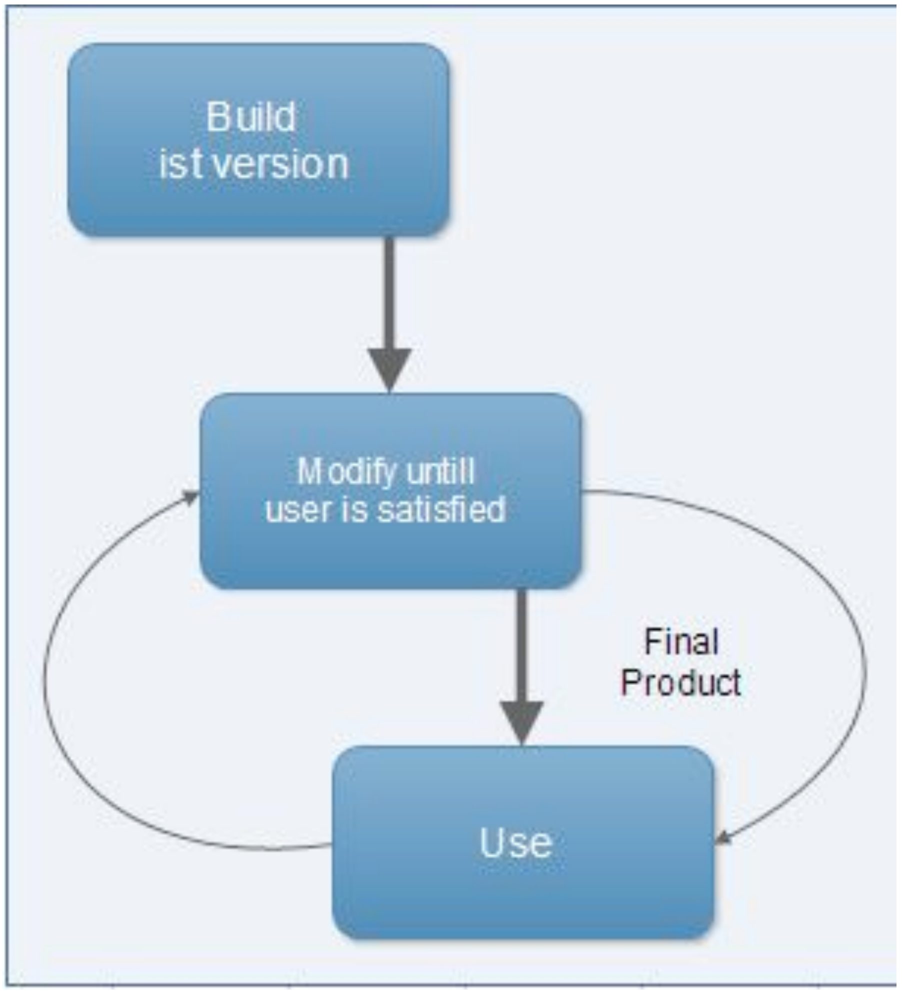
“Code and Fix” involves starting coding as soon as you have an idea of what to do.

When you encounter a bug/new requirement, you just fix it or code it up.

This is a minimalist process: **code** up what has to be done and **fix** bugs as you go. No design and no planning!

A bit like trying to get to Galway by heading West, and dealing with obstructions and challenges as they arise.

The Code and Fix Model



This resembles a primitive Agile process.

More on this later.

Can “code and fix” work?

This approach can work for a resourceful individual, but is impossible to apply when a large team is involved.

Even when a working product is achieved, further development is very challenging as the software was not developed to permit extension.

It may also be the chosen method for a small team with a very tight deadline.

Types of Systems

Bertrand Meyer proposed a useful categorisation of systems:

Acute: Failure may have a huge impact, possibly including loss of life (medical systems, transport control systems, critical banking systems, etc.).

Business: Commercial software, most web applications, games, almost all the software you use.

Casual: Student assignments, experimental software, informal websites.

A: Requires specialist treatment, not part of this module

B: The focus of this module

C: Code&Fix will suffice!

The Waterfall Process

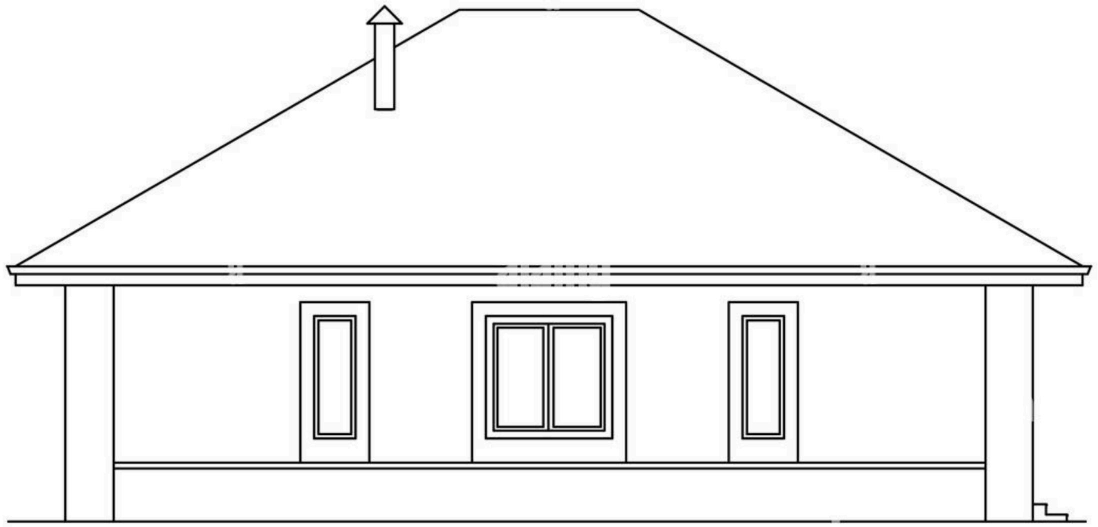
Initial attempts (~1960s) to define a software development process were based on approaches used in accepted engineering practice.

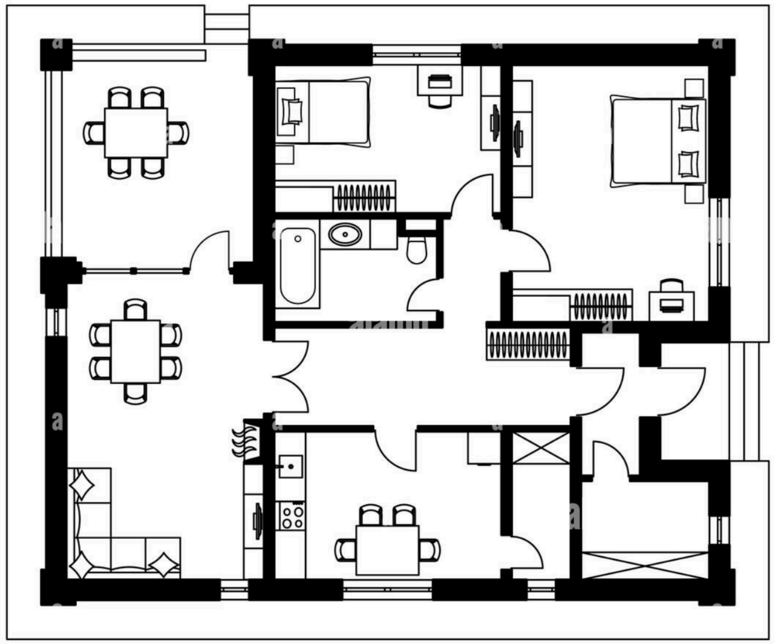
Waterfall is still in use today, and understanding it helps in grasping the now commonplace Agile processes.

Before delving into the Waterfall process, let's clarify what is meant by "accepted engineering practice."

Although the Waterfall process is largely discredited, note that in a 2018 StackOverflow developer survey, 15% of the 57,000 respondents stated that they used a Waterfall development process.

How construction works





Let's say we're building a house (or a manufacturing plant, bridge, road, rocket, etc.)

We start with high-level designs like the ones above.

Low-level designs add more detail.

Eventually we have enough detail that physical construction can begin.

How construction works

0 - Strategic definition.
1 - Preparation and brief.
2 - Concept design.
3 - Developed design.
4 - Technical design.
5 - Construction.
6 - Handover and close out.
7 - In use.

Source: RIBA

Requirements, analysis and design all happen on paper

Hard hats, boots, concrete

Passing building on to customer

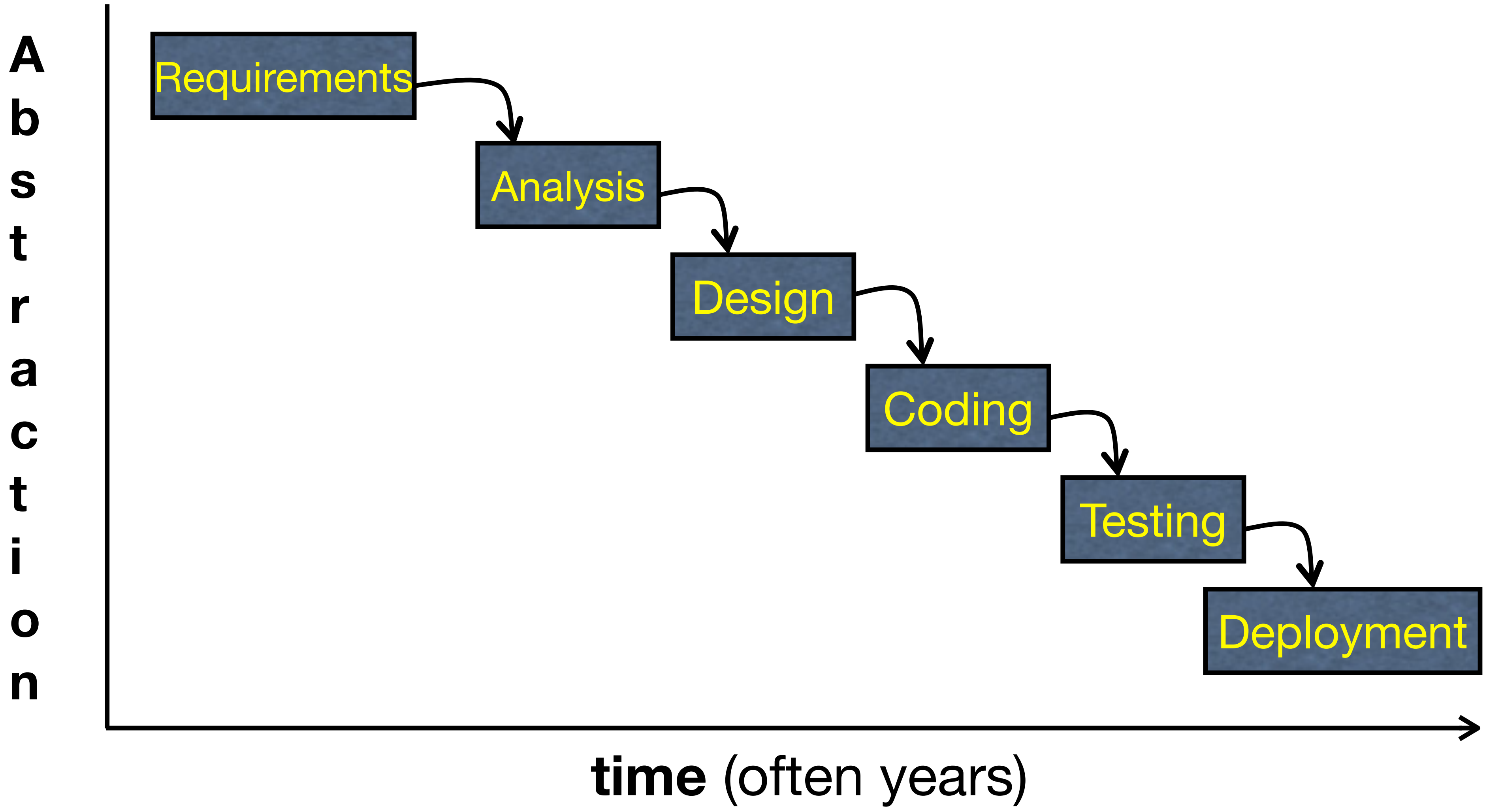
Steps 0 to 7 happen in sequence.

Steps 0 to 4 involve "blueprints" -- paper designs that are relatively easy to change. Each step adds more detail.

Once Construction is underway, changes become practically impossible.

The Waterfall Process

The Waterfall Process mimics the typical construction process we just saw.



Phases of Waterfall

Requirements: Figuring out the customer's needs.

Analysis: Building a model of ***what*** the system is to do.

Design: Building a model of ***how*** the system is to do it.

Coding: Write the code.

Testing: Test the code.

Deployment: Making the system available to customers.

Properties of the Waterfall Model

Involves a number of **distinct phases** that are completed **in order**: requirements, analysis, design, implementation, testing.

The output of each phase serves as input to the next phase (this is the “**waterfall**”).

Although appealing in many ways, the Waterfall process has several critical shortcomings.

Note: Interestingly, one of the first descriptions of the Waterfall Process [Royce, 1970] already identified many of its shortcomings.

Weaknesses of the Waterfall Model

There are three main problems with the Waterfall approach:

1. Handles changes in requirements badly.
2. Testing happens only very late in the process.
3. Real customer feedback comes only very late in the process.

Other issues include:
- Developers spend months/years not writing software
- Keeping documentation/code consistent is a burden
- Analysis->Design mapping is famously challenging
- No attention is paid to integration of software
- Human error discovered only late in the process

1. Changes in Requirements

Waterfall assumes that the requirements can be fixed at the start of the process.

In the dynamic environment where most software lives, this is an unrealistic assumption.

Changing a requirement e.g. in the design phase involves revisiting the requirements/analysis/design phases -- a lot of work.

The alternative is not to accept the new requirement, but this leads to delivering an obsolete system to unhappy customers.

2. Late Testing

The best way to uncover errors in software is:

TESTING

Testing analysis/design documents is possible, sort of...

... but testing is never more conclusive than when **testing executable code.**

In Waterfall, testing happens at the end of the process, so it is only here that errors are discovered.

Handling an analysis error at this stage involves revisiting the analysis/design/implementation phases. Tedious and time consuming.

3. Late Customer Feedback

Customers can give feedback at any stage.

However, incisive feedback only comes when they get to use the system.

In Waterfall, this only happens when the system is finally deployed.

This is far too late in the process, and the typical customer response is something like:

"*Oh, this isn't what I expected at all...*"

Analysis Paralysis

The problems outlined often led to waterfall projects suffering from what is jokingly called **analysis paralysis**.

The Analysis has to be perfect before Design starts. Perfection is impossible, so progress is halted.

With the project deadline looming, the analysis&design was typically ditched and the developers just start coding.

The result was little better than code-and-fix.

Improving on Waterfall

Current software development processes aim to manage change and human error better, and place a heavy emphasis on early deployment and testing.

Two key features of any contemporary software process are:

Incremental
Iterative

We'll look at these properties separately before considering concrete development processes that are based on them.

(Although they are distinct properties, they go very much hand-in-hand)

Incremental Development

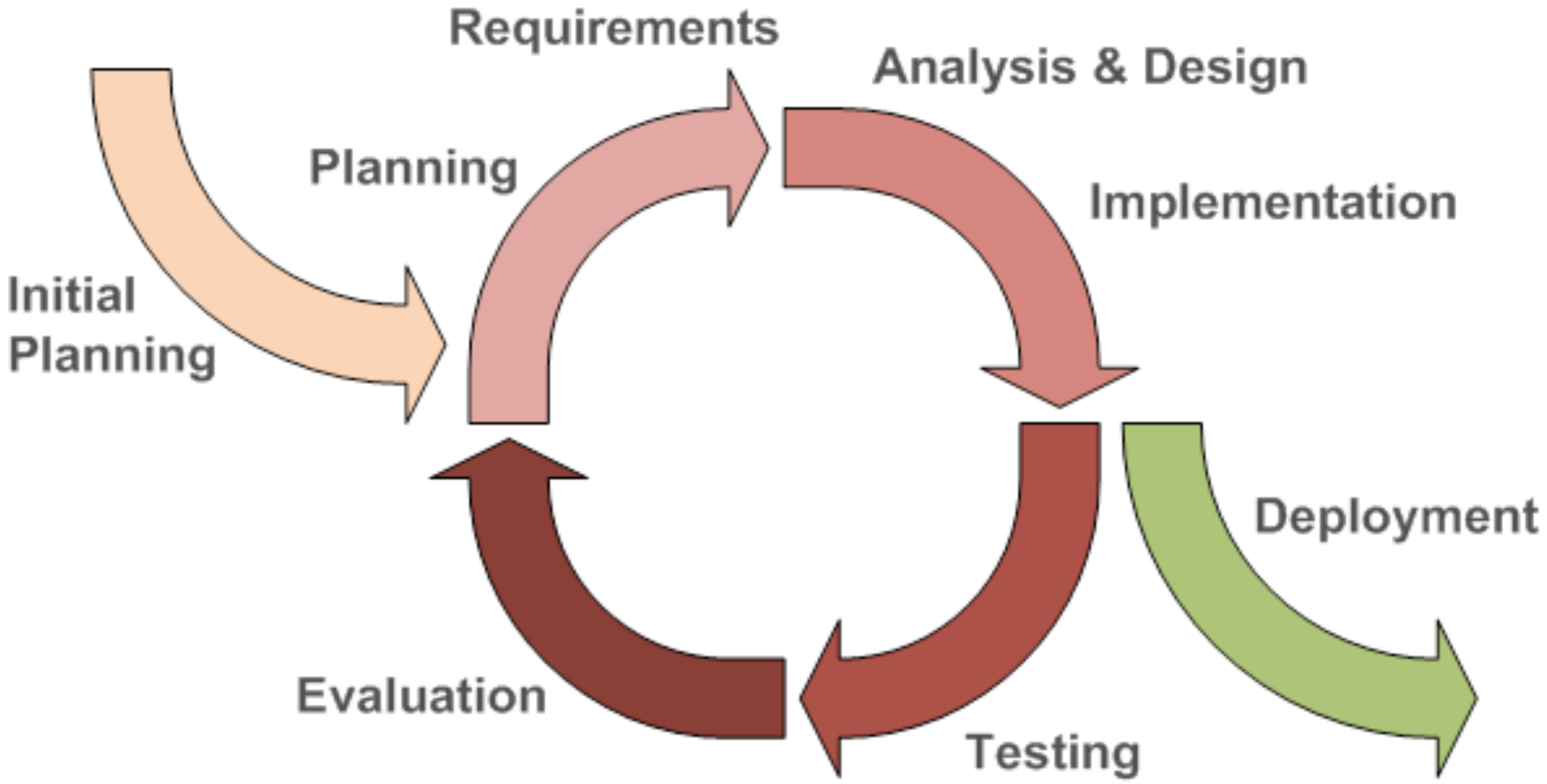
In an **incremental** (or evolutionary) process, a simple working prototype is initially produced and this is gradually extended (evolved) to the final system.

Advantages of incremental development:
- Early testing and feedback from customers
- Easy integration of new requirements

Risks of incremental development:
- Hard to manage -- will the increments ever result in the final system?
- How is the overall system design developed?

Iterative Development

The key property of an **iterative** model is that several passes through the various stages of the software process are made, e.g.



Iterative vs. Incremental Development

Iterative and *Incremental* are distinct properties.

It is possible (in theory) to have one without the other.

In practice they always go together.

As we'll see, contemporary software processes (Agile processes) are invariably both iterative and incremental.

BDUF

Big **D**esign **U**p **F**ront -- a pejorative term used by Agilists to refer to Waterfall-like processes.

Joel Spolsky* wrote on BDUF (in 2005):

> *I can't tell you how strongly I believe in BDUF... I'm proud to use it, no matter what the XP fanatics claim. They're just wrong on this point and I can't be any clearer than that.*

It all depends on context. Full Waterfall is rarely appropriate, but BDUF/LDUF definitely has its uses.

* Joel Spolsky: Creator of Trello and StackOverflow.

What is *Software Maintenance?*

Common definition:"Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs."

What percentage of software development falls under this definition?

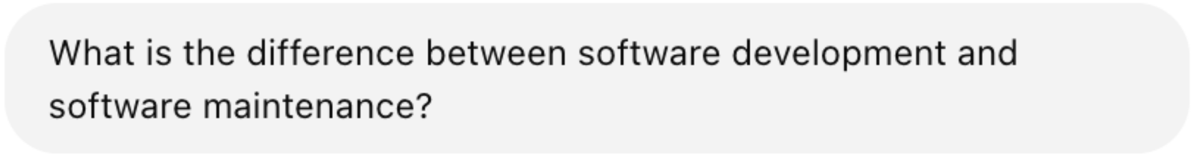
Traditional answer: ~70%

The truth is closer to 99%

The vast majority of software developed these days falls under the "maintenance" definition.

i.e. development ~= maintenance

Let's ask ChatGPT!


What is the difference between software development and software maintenance?

Partial Answer (abridged):

Development is the initial creation of the software. It involves:
- Planning & Requirements Gathering
- Designing the software
- Implementation (Coding)
- Testing: Ensuring the software works as expected.
- Deployment: Releasing the software for use.

This answer is wrong (as expected).

It's based on the Waterfall process, which was largely discarded by the industry in the early part of this century.

Beware of using web sources, or AI tools, for this module!

Agile Processes

See separate slide deck
for this topic.

Snapshot of Irish Software Industry

Searching LinkedIn IT jobs in Ireland for terms related to software processes:

	Jan'20	Jan'21	Jan'23	Jan'24	Jan'25
"**Agile**"	3,006	3,033	6,583	2,382	1,826
"**Scrum**"	965	984	2,821	1,459	1,283
"**Waterfall**"	185	162	291	23	**?**

Not a definitive measure of course, but gives an idea of the popularity of these techniques.

Final Remarks

Software Development Process is a dynamic area where research struggles to uncover any generalisable conclusions.

In this type of space, advocacy flourishes and you'll find many "influencers" online promoting their own perspective.

Most are worth listening to; none have the whole truth.

Summary

The process we use for developing software is a critical issue, and there is no definitive best approach.

There has been an evolution both in terms of the metamodel used and development process used.

The object-oriented metamodel is dominant in the software industry.

Most current software processes are Agile in nature, including Scrum, Extreme Programming, Kanban and others.