Introduction to Software Engineering

Introduction

How well are we doing?

The Standish Chaos Report is often quoted. The Standish Group did a major survey of software development projects way back in 1994.

The results were a shock to the industry at the time. The surveys have been performed at roughly 2-year intervals since then, and now yearly.

The 1994 Report found:

- Only 16% of development projects in the USA were completed on-time, on-budget, and with all the originally planned features.

- 31% were *cancelled* before they were completed.

- The remaining 53%, on average:
  - *overran* their original cost estimate by 89%,
  - *exceeded* their original schedule by 122%,
  - provided only 61% of the *features* in the original specification.

CSIS0297/COMP3297: Introduction to Software Engineering

Chaos Report: Traditional Resolution Definition

Focus is on quality of development. Classifies software projects as:

Successful
- Completed on time and within budget, offering all features and functions as initially specified
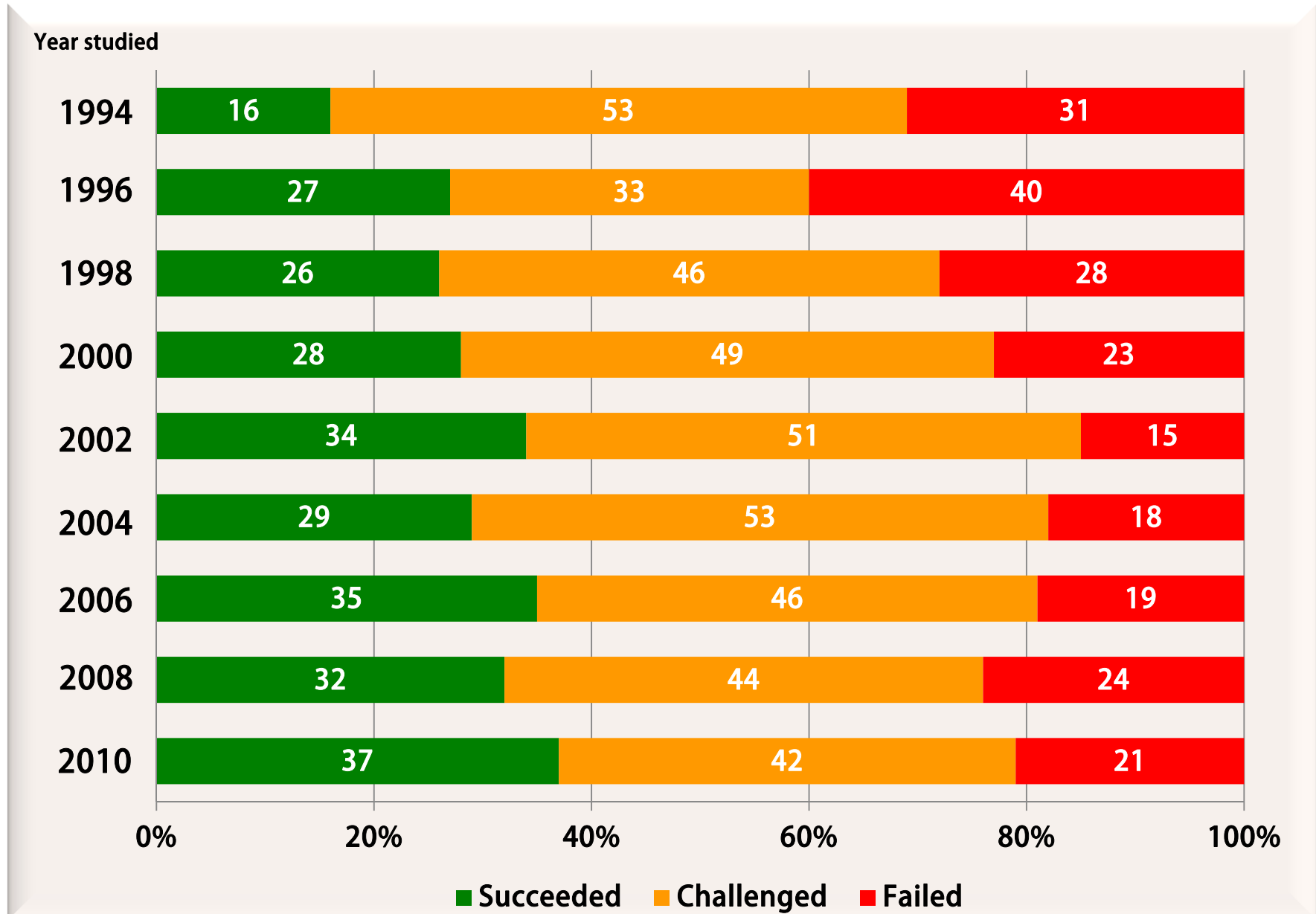
Challenged
- Completed and operational but over budget and over time estimate, and offers fewer features and functions than originally specified

Failed
- Cancelled at some point during development or never used

All CHAOS Reports to 2010: Standish Group

Year studied

Year	Succeeded	Challenged	Failed
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2002	34	51	15
2004	29	53	18
2006	35	46	19
2008	32	44	24
2010	37	42	21

■ Succeeded   ■ Challenged   ■ Failed

Chaos Report: Modern Resolution Definition

Should we really judge "success" in terms of delivering the system *exactly* as specified at the beginning of the project?
Following criticism, Standish changed the definition

Successful

- Completed on time and within budget, *with a satisfactory result*. Here the different definition of success reflects how teams were actually judged by clients: for example "solution was delivered and met its success criteria within a range acceptable to the organization"
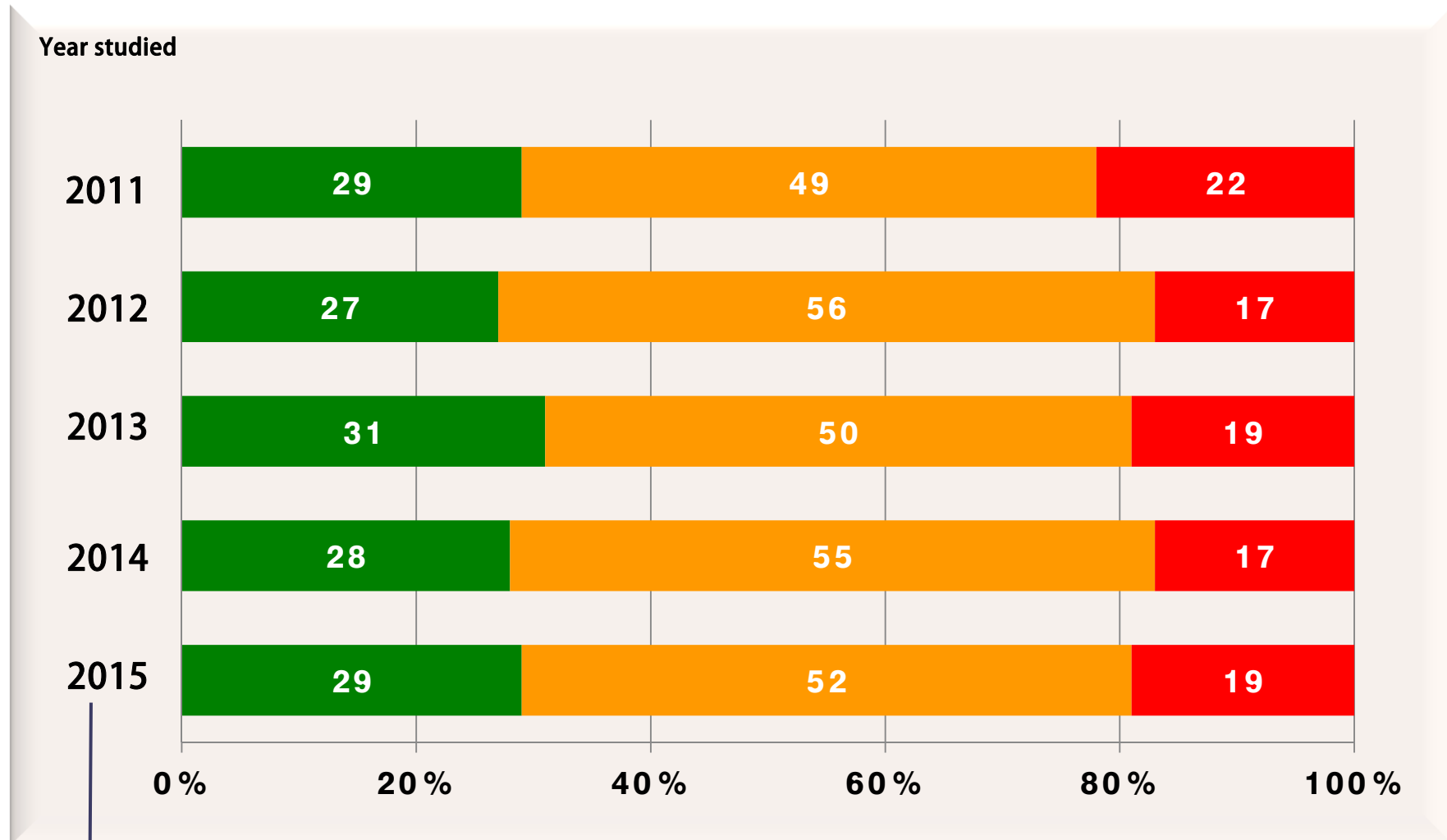
Challenged

- Completed and operational but over budget, late, with unsatisfactory results

Failed

- Cancelled at some point during development or never used

Recent CHAOS Data: Modern Resolution

Year studied

Year	Green	Orange	Red
2011	29	49	22
2012	27	56	17
2013	31	50	19
2014	28	55	17
2015	29	52	19

0 %   20 %   40 %   60 %   80 %   100 %

50,000 projects worldwide, from small enhancements to massive re-engineering

Large projects are particularly risky

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

The CHAOS reports have been questioned, but other data have pointed in a similar direction.

Dynamic Markets Survey, 2007 (8 countries)

- 62% of projects overrun on time.

- 49% of projects overrun on budget.

- 47% of projects suffer higher than expected maintenance costs.

- 28% of organizations have experienced projects that do not fit requirements.

- 25% of organizations have seen business users reluctant to adopt new systems.

- 16% of organizations reported projects having a negative effect on existing systems.

- 13% of organizations say projects have not delivered expected ROI.

Main enemy 1: Complexity

- Large systems are complex – many components must interact to provide the functionality required by users

- The problem the system is required to solve (the required functionality) is usually complex too

- Large systems are developed in teams. This requires communication and coordination which adds even more complexity to the process

Our approach to development must handle complexity:

- in the system being developed

- in the development process itself

Main enemy 2: Change

- Large systems take a long time to develop

  - their environment and the needs of stakeholders are likely to change during development

  - if ignored, it can happen that development is executed "perfectly" but the product is a failure

- Large systems have a long lifetime

  - can expect a lot of modification during that lifetime in order to adapt to new requirements or new technology

Our approach to development must handle constant change:

  - **during initial development, and**

  - **after delivery**

Software Engineering Solution

Definitions of Software Engineering capturing some of what we are looking for

An early definition − captures the ideas of complexity and change, but nothing about engineering discipline

"Multi-person construction of multi-version software"   Parnas

IEEE Definition of Software Engineering

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

But neither mentions the goal of all engineering disciplines: *"building high quality, cost-effective solutions to practical problems"*

Our working definition

Software Engineering is the process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints and in the context of constant change.

Lethbridge

Focus on customers' needs

Don't lose sight of the
true objective

Software engineering is not a purely technical discipline.

We need to make sure we build the *right system*

Software Engineering is the process of *solving customers' problems* by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints and in the context of constant change.

The engineering part:
raises importance of process

This is what makes it *"engineering"*.
Although the discipline is still developing, there are many well-accepted practices.

These are combined to form effective *software process*.

Software Engineering is the process of solving customers' problems by the *systematic* development and evolution of large, high-quality software systems within cost, time and other constraints and in the context of constant change.

Acknowledge the importance of maintenance

Software Engineering is the process of solving customers' problems by the systematic development and *evolution* of large, high-quality software systems within cost, time and other constraints and in the context of constant change.

60%-80% of total development effort is maintenance after the product is first delivered

Large scale development

Software Engineering is the process of solving customers' problems by the systematic development and evolution of *large*, high-quality software systems within cost, time and other constraints and in the context of constant change.

Small scale programming is a solitary activity.

Large-scale development is very different and requires cooperation between developers whose *combined* work products result in the final system

Engineering Software

**Structuring Software Development:
Introduction to Software Process**

What is a software process?

A set of activities and the way those activities are structured to develop a software product

This is systematic, but not to the level where it can be automated. Certainly less systematic than other branches of engineering.

Software processes are complex and involve a lot decision-making. There is no single concrete recipe that guarantees success.

Success still requires creativity and judgement.

Common development activities

The *process* defines activities and how they are organized into a project.

The objective of the project is to transform a user's needs into a software system.

Generally we can separate activities into:

- Management Activities that span the entire project, and
- Technical Activities

But while software development projects have many activities in common, that does not mean that all projects should be organized in the same way.

Software Development Activities

Requirements Engineering

- Domain analysis
- Problem definition
- Requirements gathering
- Requirements analysis
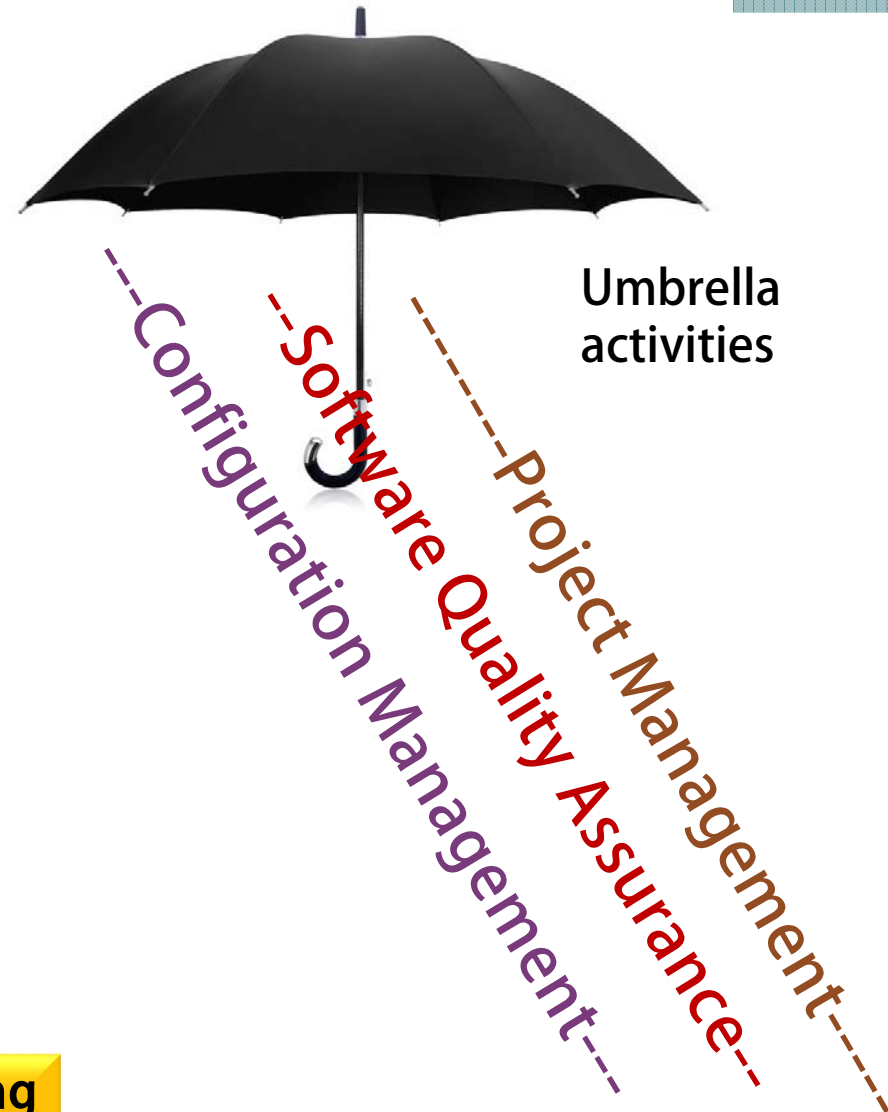- Requirements specification

Design

- [Systems engineering]
- Software architecting
- Detailed design
- User interface design
- Database design

Implementation

Testing

Deployment

Maintenance

Umbrella activities

--Configuration Management--

--Software Quality Assurance--

---Project Management------

Software Process

During a project, the process answers questions such as:

> What do we do next?
> For how long do we do it?
> What do we produce as a result?

intermediate and final products – *artefacts*

We'll see there are many different *Process Models* that provide general frameworks for creating this organization.
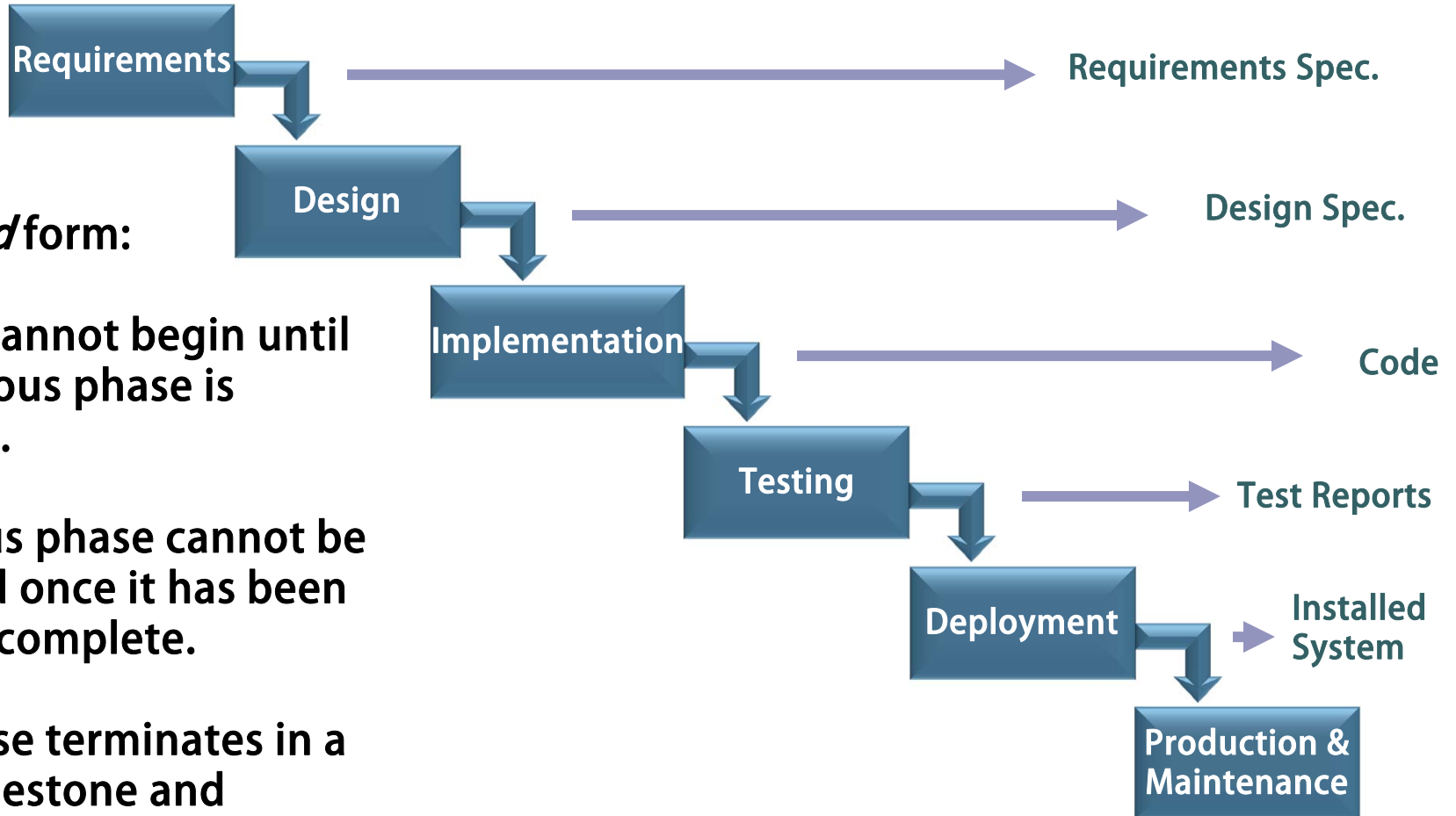
Q: What hangs in the framework?

A: Software development activities.

The simplest (but poor) model is one in which the stages of the model directly reflect the basic development activities: the Waterfall Model.

Waterfall Model

Traditional software engineering
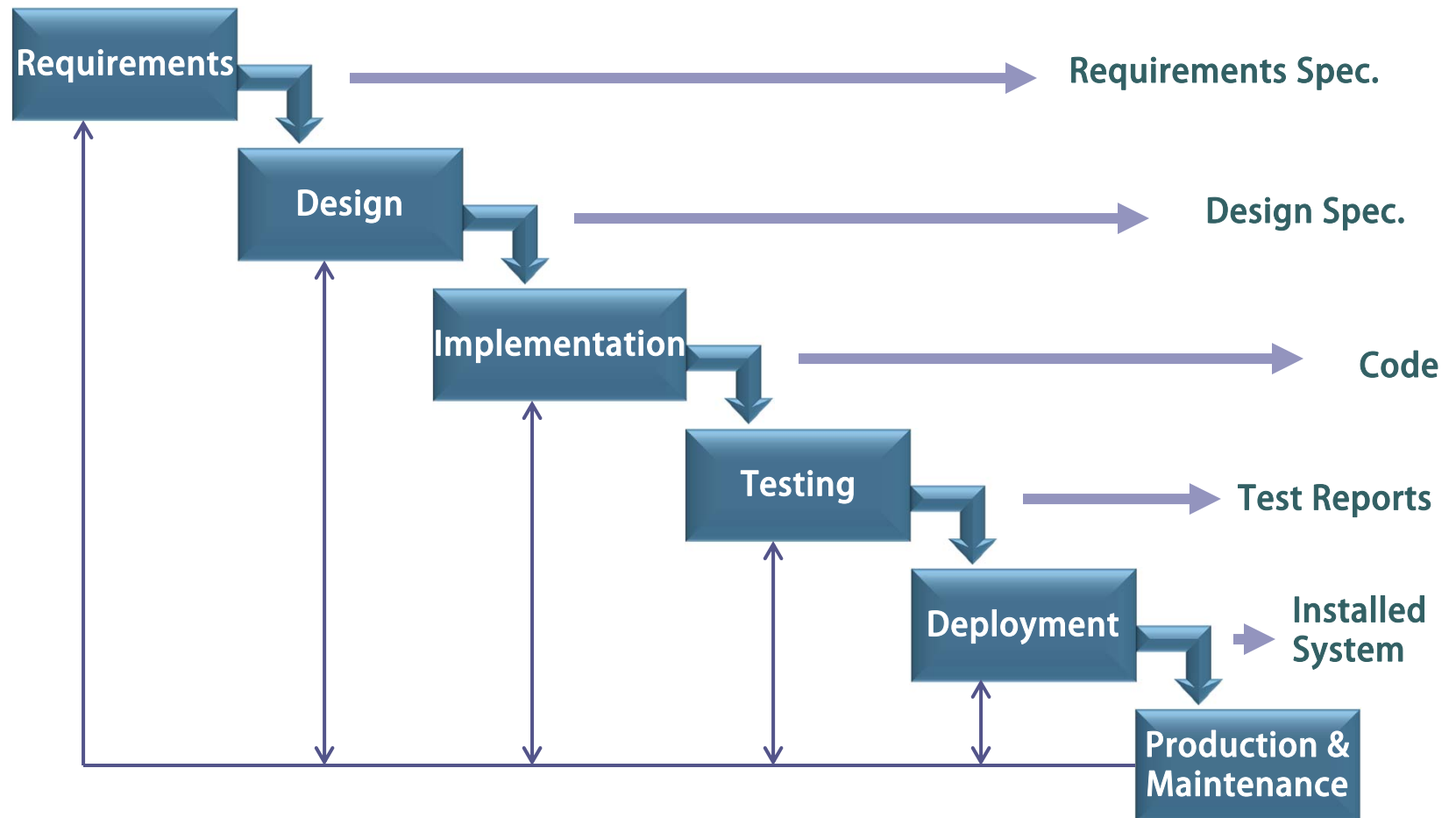
Typical Deliverables (also milestones)

In its *rigid* form:

A phase cannot begin until the previous phase is complete.

A previous phase cannot be reentered once it has been declared complete.

Each phase terminates in a major milestone and deliverable. A review follows each phase.

Requirements → Requirements Spec.

Design → Design Spec.

Implementation → Code

Testing → Test Reports

Deployment → Installed System

Production & Maintenance

Waterfall Model with Feedback

In practice, there is feedback as problems are discovered in later phases. This involves so much expensive rework of documents that they are typically frozen after limited feedback.

Major problems with Waterfall

Easy to manage, but:

- Can lead to badly structured systems because of workarounds needed for prematurely frozen designs.

- Inflexible. Discourages change - volume of new documents and repeated steps makes change expensive. Commitments are made early and project can't respond to changing requirements or new information.
  Can only work when requirements are well understood and unlikely to change during development, and risks are low.

So, what sort of project could this be?

- Customer doesn't see a working program until late in the project. A mistake in requirements analysis can easily remain undetected for a long period − expensive to fix.

Relative cost of fixing a requirements defect at various stages of development
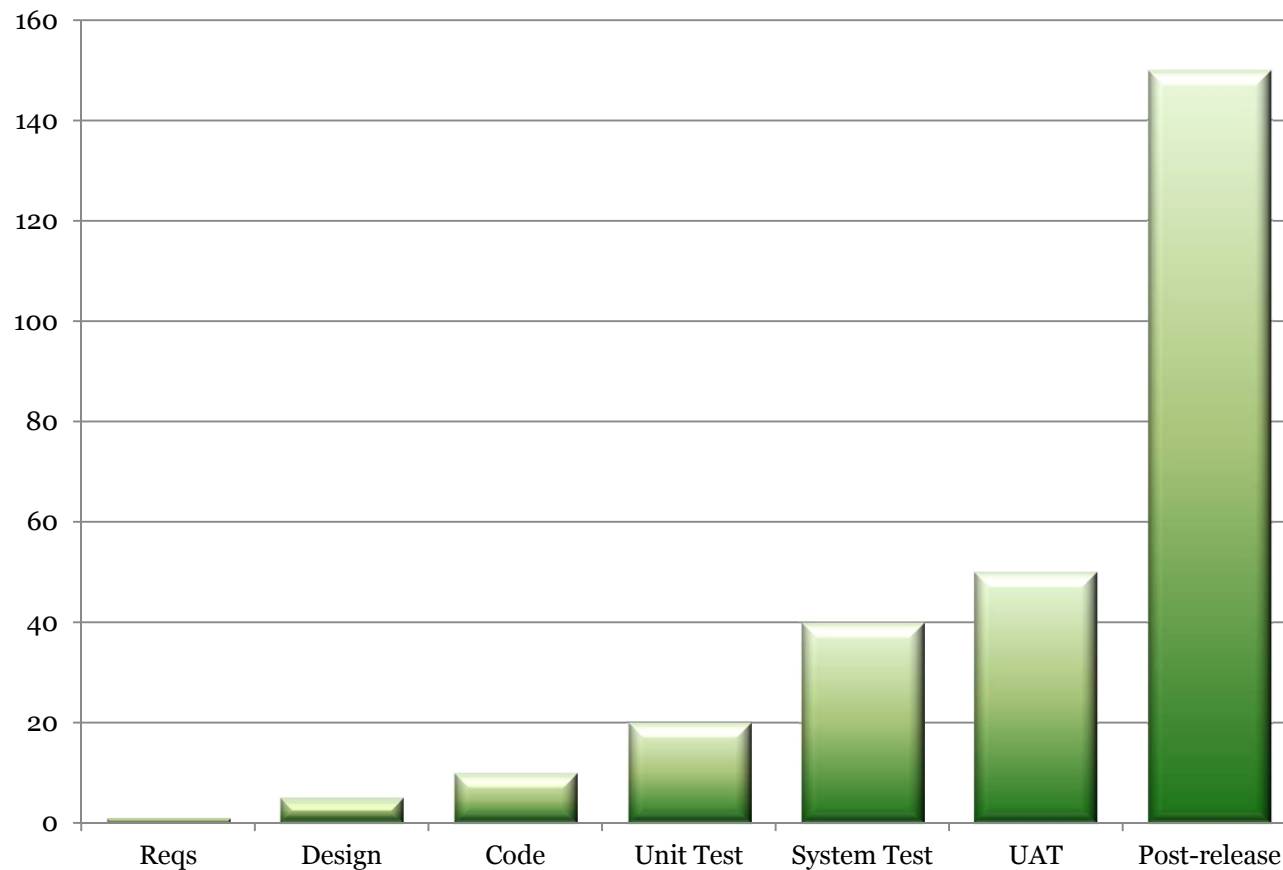
A better approach: Evolutionary Models

Requirements usually change over the course of system development.

So, the software product under development must evolve over time if we want to satisfy those requirements.

Can accommodate change by developing the core of the application providing essential features. Then refine and extend incrementally.
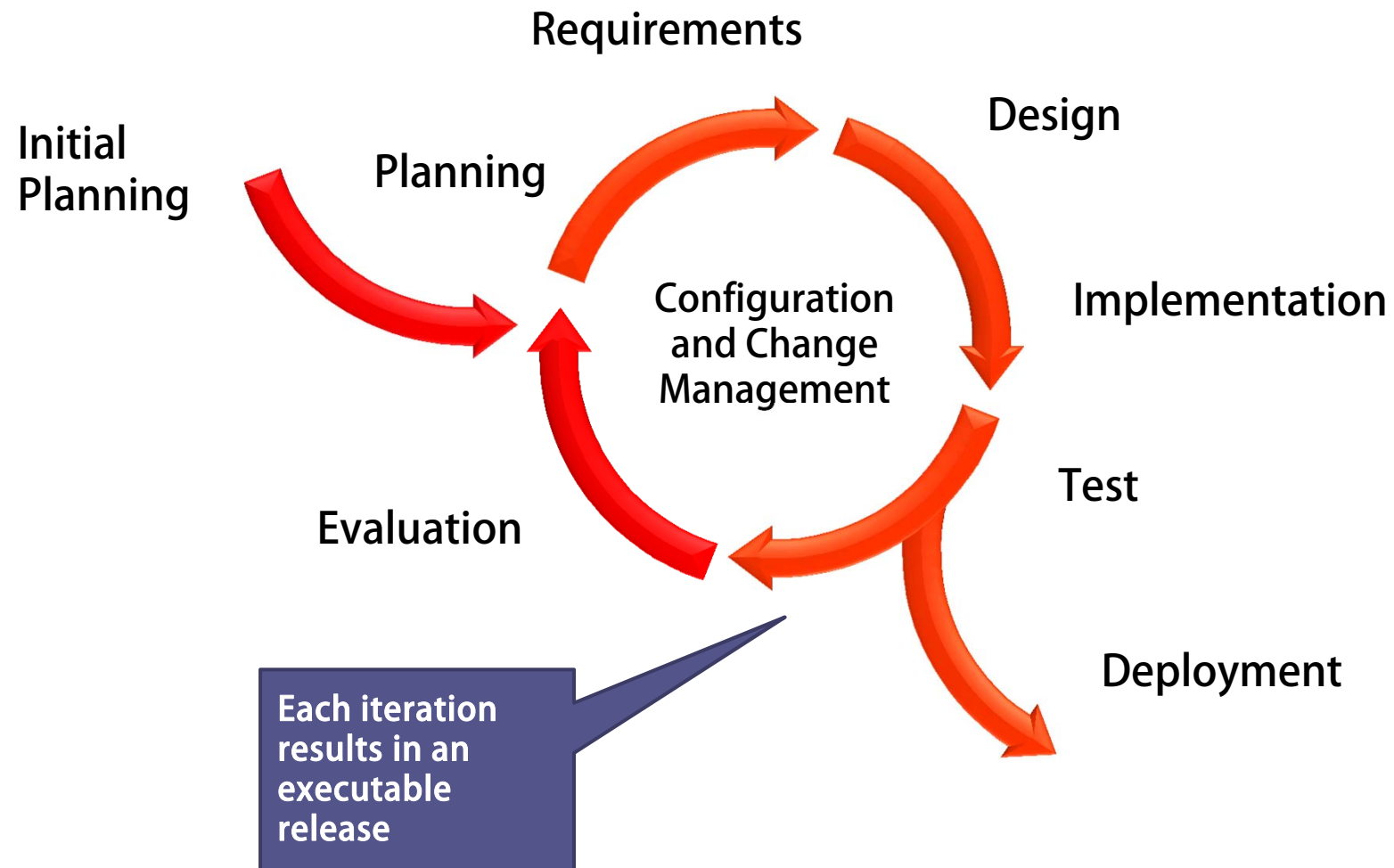
Development proceeds *iteratively* and *incrementally*

Iterative and Incremental Process

Simple idea. Develop in a sequence of small, self-contained projects − the *iterations*

- Each iteration has well-defined goals, but these are not set until planning *at the start of the iteration*

- A deliverable of each iteration is a *working* release − a stable, integrated, tested, partially complete system. This is *proof* that the iteration goals were achieved can be evaluated by customers

- An iteration builds on the product of the previous iteration. So the system grows *incrementally,* and is a working product at every stage.

Additional advantage: In a competitive market, users often can't wait for a *fully* functional system to be developed.

The iterative cycle

Requirements

Initial
Planning

Planning

Design

Configuration
and Change
Management

Implementation

Evaluation

Test

Each iteration
results in an
executable
release

Deployment

What do we do first?
What do we do next?

The CHAOS Reports show that software development is often a very risky undertaking

The question we need to answer for every project:

How can we best use the iterative and incremental approach to handle risk?

We'll examine this in our first class exercise.