Software Engineering and Best Practices

Sources: Various.
Rational Software Corporation slides,
OOSE textbook slides, Per Kroll talk, How to Fail with the RUP article, textbooks
Most slides have been modified considerably

Fundamental Terms / Concepts

- Science and Engineering
  - Discover
    - Relationships that exist but are not found
    - Formulas; chemical composition, d=r*t; calories in fats, carbohydrates, proteins; experimentation;
    - Astrophysics – origins of the universe
  - Build
    - Apply principles of science and mathematics to real needs, commodities, structures, products, etc.
- Software Engineering; Software Development

Fundamental Concepts / Terms (2)

- Software Engineering; Software Development
- Job positions:
  - Software developer
  - Programmer
  - Software engineer
  - Analyst / Programmer
  - Senior … what have you…

What is Software Engineering?

- The process of solving customers’ problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints
- Note:
  - Process, systematic (not ad hoc), evolutionary…
  - Constraints: high quality, cost, time, meets user requirements

Analysis of the Definition:

- Systematic development and evolution
  - An engineering process involves applying well understood techniques in a organized and disciplined way
  - Many well-accepted practices have been formally standardized
    - e.g. by the IEEE or ISO
  - Most development work is *evolutionary*

- Large, high quality software systems
  - Software engineering techniques are needed because large systems cannot be completely understood by one person
  - Teamwork and co-ordination are required
  - Key challenge: Dividing up the work and ensuring that the parts of the system work properly together
  - The end-product that is produced must be of sufficient quality

- Cost, time and other constraints
  - Finite resources
  - The benefit must outweigh the cost
  - Others are competing to do the job cheaper and faster
  - Inaccurate estimates of cost and time have caused many project failures

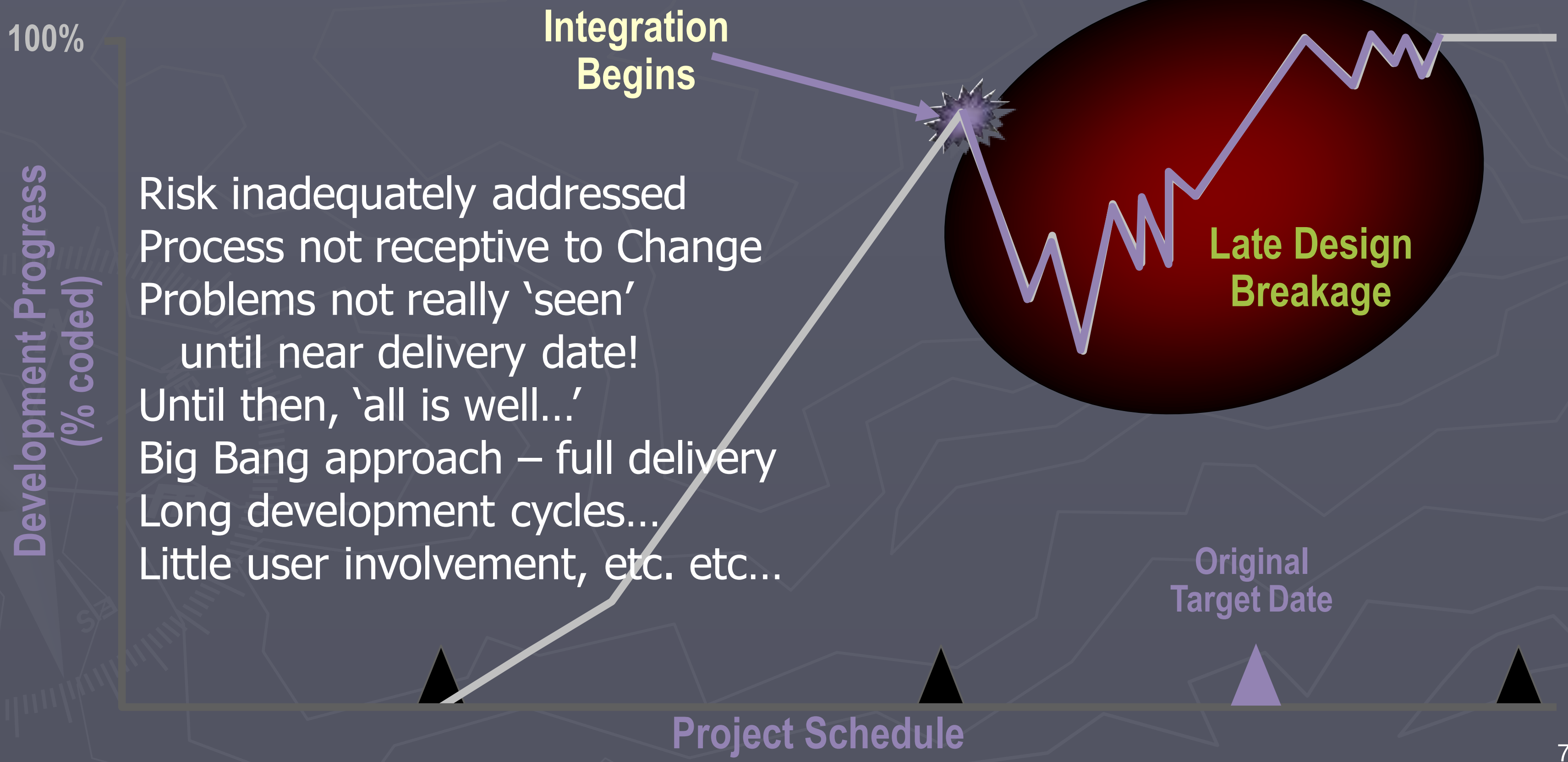
Comments:

- $250 billion annually in US.
- Over 175,000 projects!
- Complexity, size, distribution, importance push our limits.
- Business pushes these limits:
  - Great demands for rapid development and deployment
- → Incredible pressure: develop systems that are:
  - On time,
  - Within budget,
  - Meets the users' requirements
- Figures in the late 90s indicated that at most
  - 70% of projects completed
  - Over 50% ran over twice the intended budget
  - $81 billion dollars spent in cancelled projects!!
- Getting better, but we need better tools and techniques!

What Happens in Practice

Sequential activities: (Traditional 'Waterfall' Process)

Requirements → Design → Code → Integration → Test



Risk inadequately addressed
Process not receptive to Change
Problems not really 'seen'
until near delivery date!
Until then, 'all is well…'
Big Bang approach – full delivery
Long development cycles…
Little user involvement, etc. etc…

Symptoms of Software Development Problems

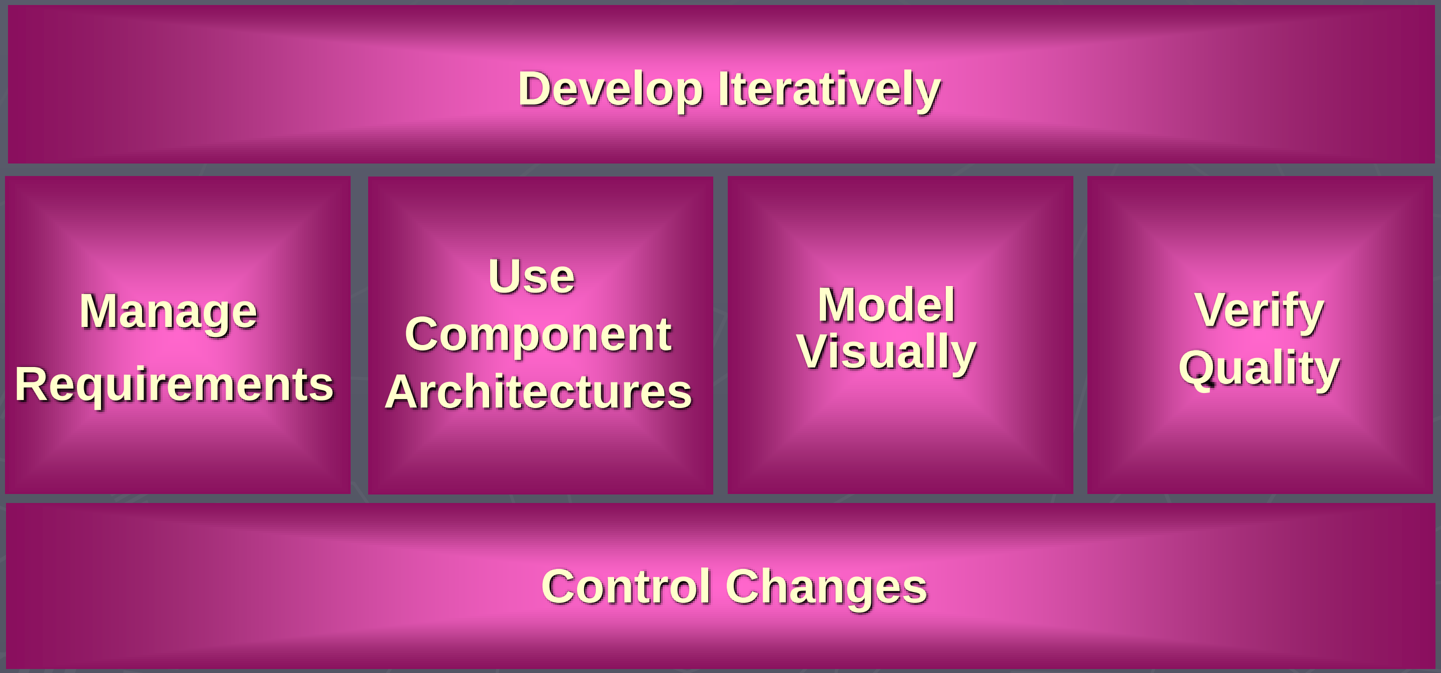
- Inaccurate understanding of end-user needs
- Inability to deal with changing requirements
- Modules that don't fit together (integration)
- Software that's hard to maintain or extend (brittle)
- Late discovery of serious project flaws (integration)
- Poor software quality (architecture, risks unanticipated…)
- Process not responsive to Change (Gantt Charts…)
- Unacceptable software performance
- Team members in each other's way, unable to reconstruct who changed what, when, where, why (software architecture, …
- …and we could go on and on…

Need a Better Hammer!

- We need a process that
  - Will serve as a framework for large scale and small projects
  - → Adaptive – embraces ‘change!’
    - Opportunity for improvement not identification of failure!
  - Iterative (small, incremental ‘deliverables’)
  - Risk-driven (identify / resolve risks up front)
  - Flexible, customizable process (not a burden; adaptive to projects)
  - Architecture-centric (breaks components into ‘layers’ or common areas of responsibility…)
  - **Heavy** user involvement

- Identify best ways of doing things – a better process – acknowledged by world leaders…

Best Practices of Software Engineering



Develop Iteratively

Manage Requirements

Use Component Architectures

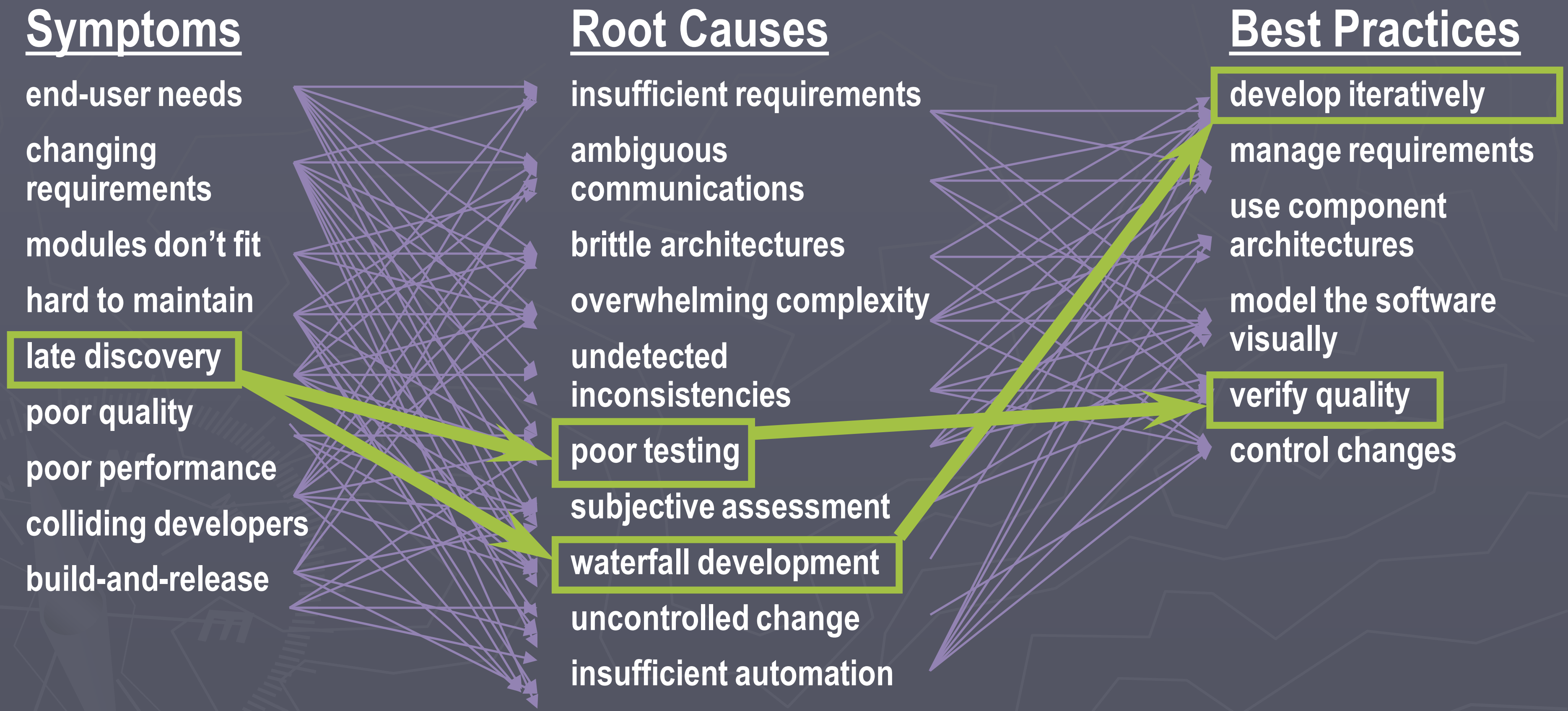
Model Visually

Verify Quality

Control Changes

Know these!

Addressing Root Causes Eliminates the Symptoms



Symptoms

- end-user needs
- changing requirements
- modules don't fit
- hard to maintain
- late discovery
- poor quality
- poor performance
- colliding developers
- build-and-release

Root Causes

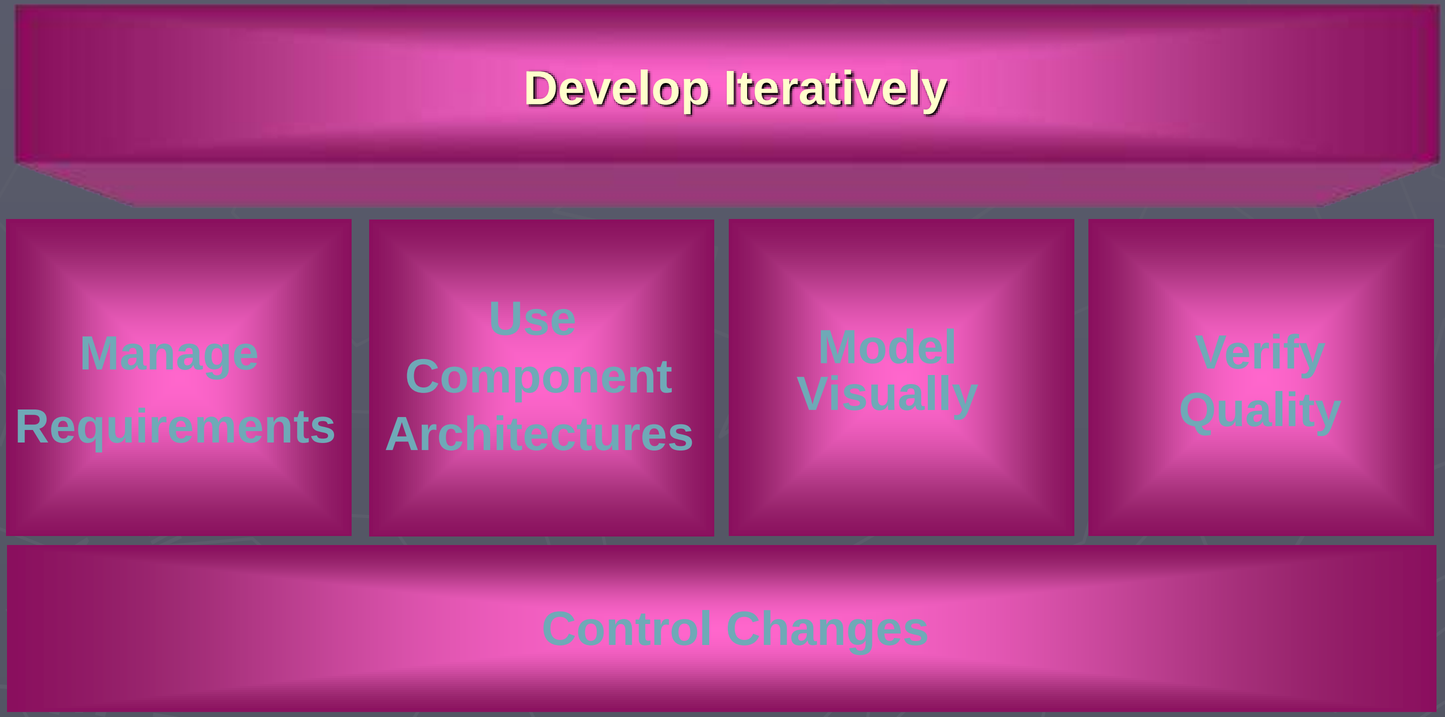
- insufficient requirements
- ambiguous communications
- brittle architectures
- overwhelming complexity
- undetected inconsistencies
- poor testing
- subjective assessment
- waterfall development
- uncontrolled change
- insufficient automation

Best Practices

- develop iteratively
- manage requirements
- use component architectures
- model the software visually
- verify quality
- control changes

Symptoms of problems can be traced to having Root Causes.
Best Practices are 'practices' designed to address the root causes of software problems.

Practice 1: Develop Software Iteratively



Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually

Verify Quality

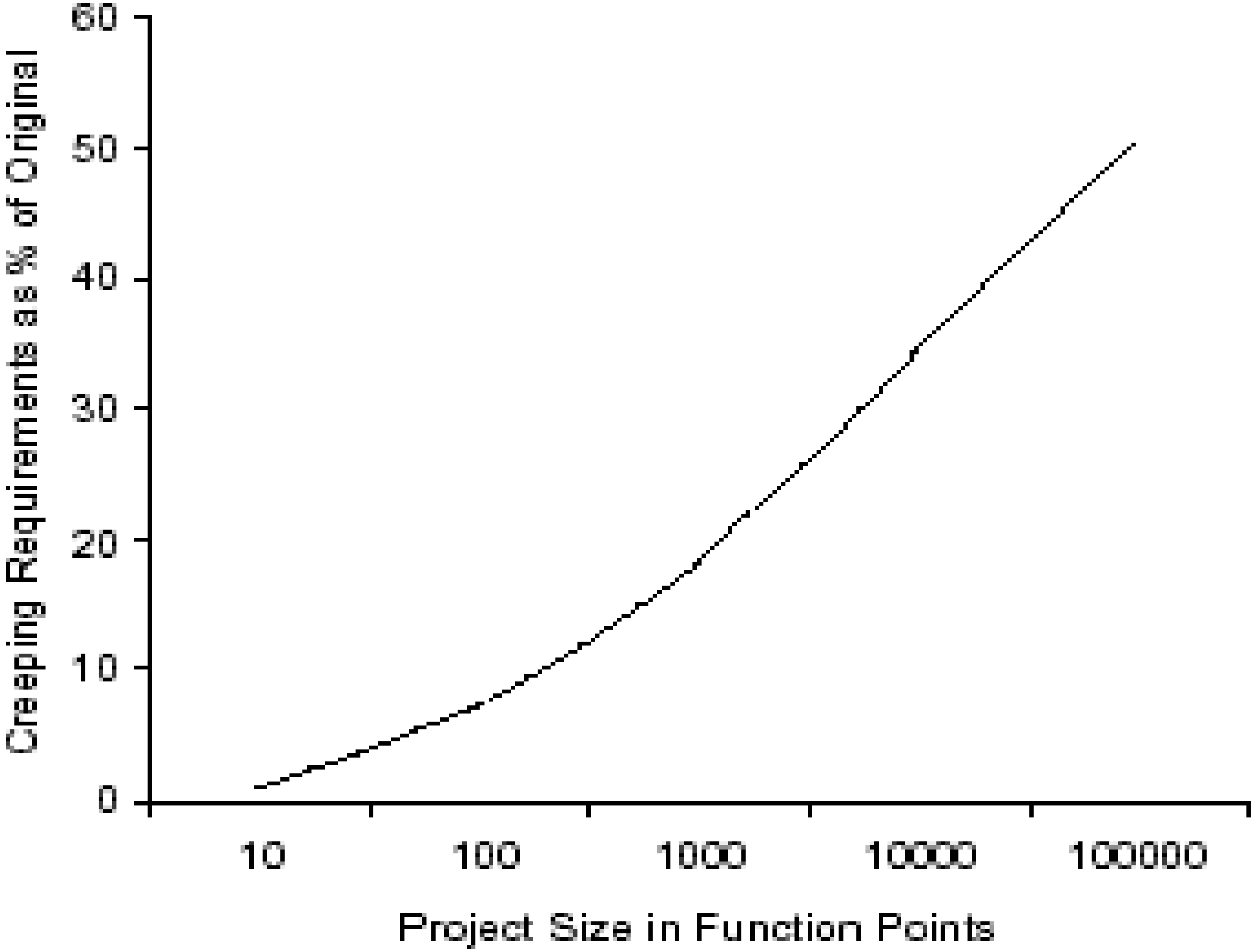
Control Changes

Considered by many practitioners to be the most significant of the six

Practice 1: Develop Software Iteratively

- Until recently, developed under assumption - most requirements can be identified up front.

- The research deconstructing this myth includes work by Capers Jones. (See next slide) In this very large study of 6,700 projects, creeping requirements — those not anticipated near the start—are a very significant fact of software development life, ranging from around 25% on average projects up to 50% on larger ones.



➔ Look up a definition of ‘Function Points.’

Figure 1 Changing Requirements are the Norm

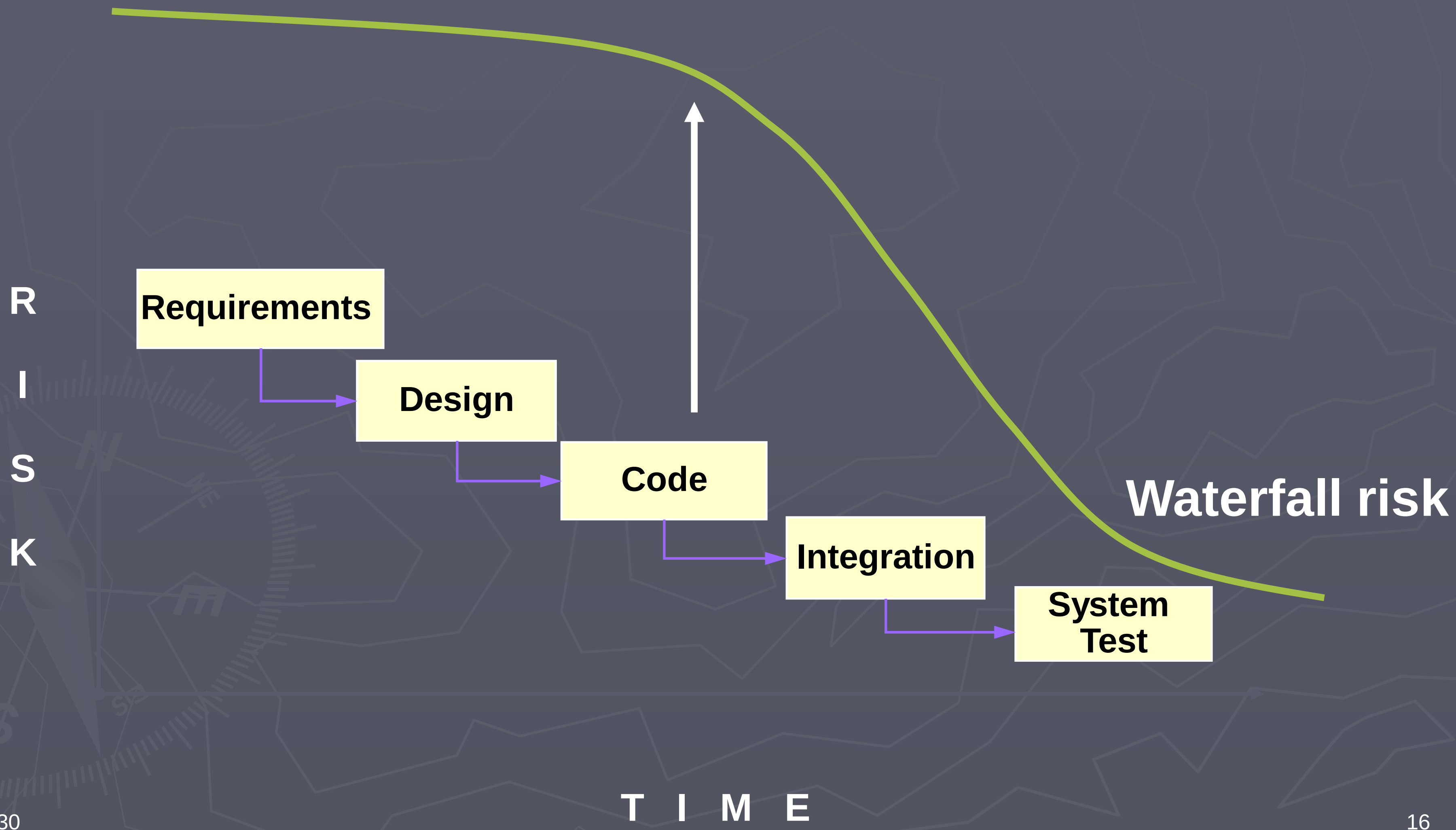
Interestingly,

- An initial design will likely be flawed with respect to its key requirements. Requirements rarely fully known up front!
- Late-phase discovery of design defects results in costly over-runs and/or project cancellation
  - Oftentimes requirements change – even during implementation!

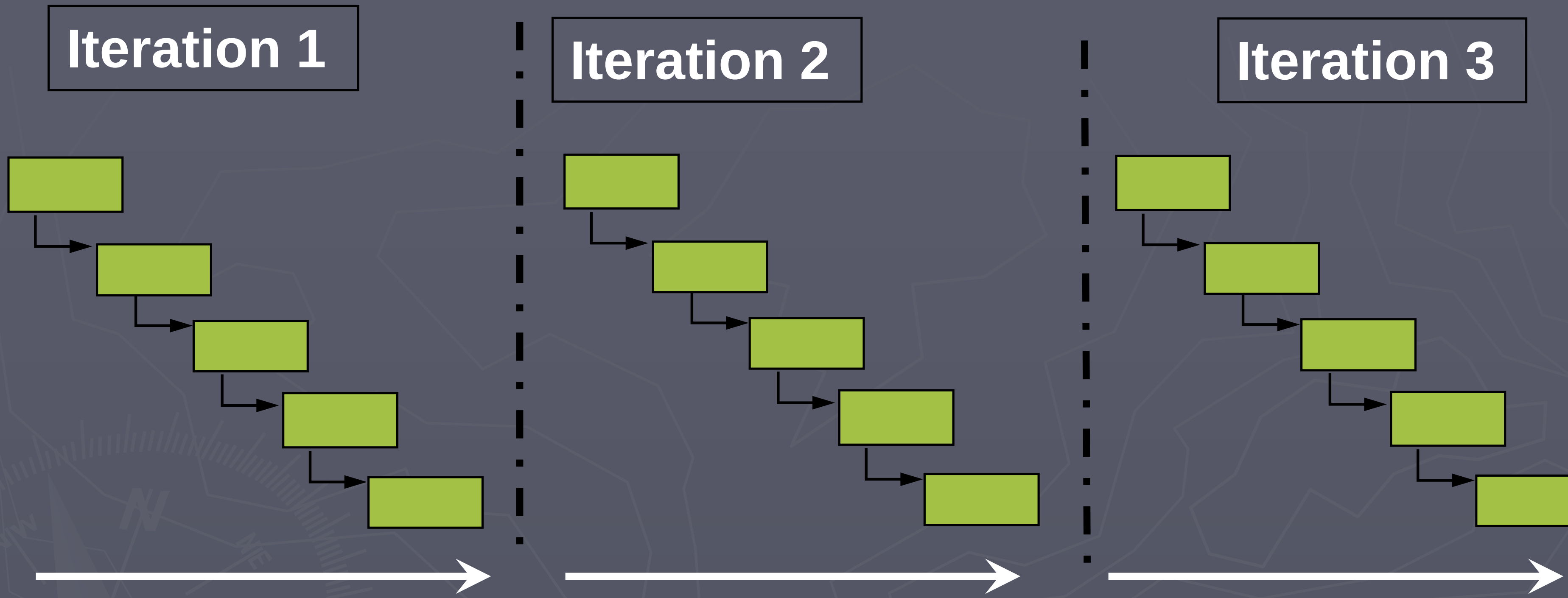
- While large projects are more prone to cost overruns, medium-size/small projects are vulnerable to cancellation.
- The key reasons continue to be
  - poor project planning and management,
  - shortage of technical and project management expertise,
  - lack of technology infrastructure,
  - disinterested senior management, and
  - inappropriate project teams.”

Waterfall Delays Risks

Walker Royce, 1995



Iterative Development



Iteration 1

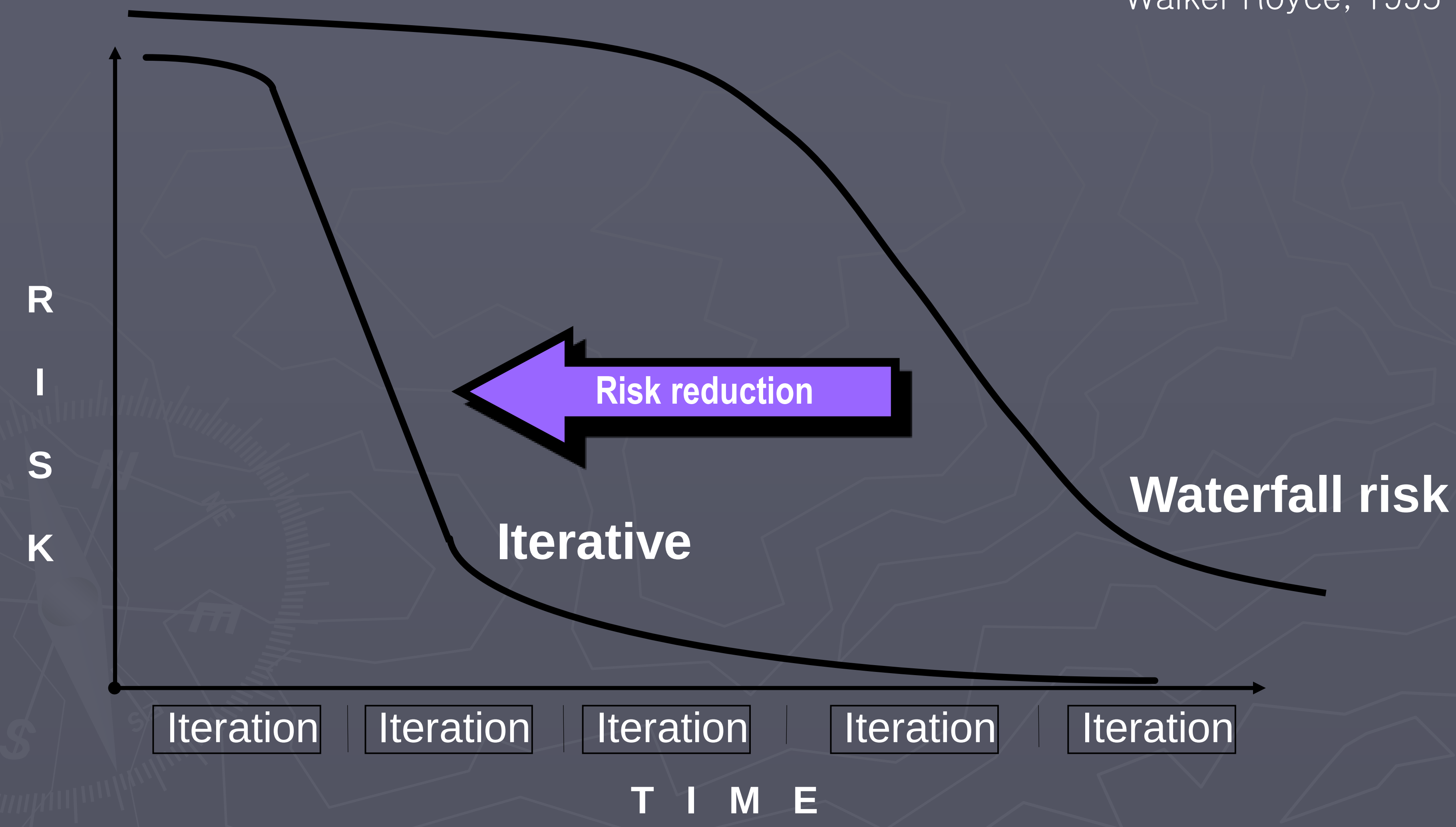
Iteration 2

Iteration 3

- ➔ Earliest iterations address greatest risks
- Each iteration produces an executable release
- Each iteration includes integration, test, and assessment!
- Objective Milestones: short-term focus; short term successes!

Accelerate Risk Reduction

Walker Royce, 1995

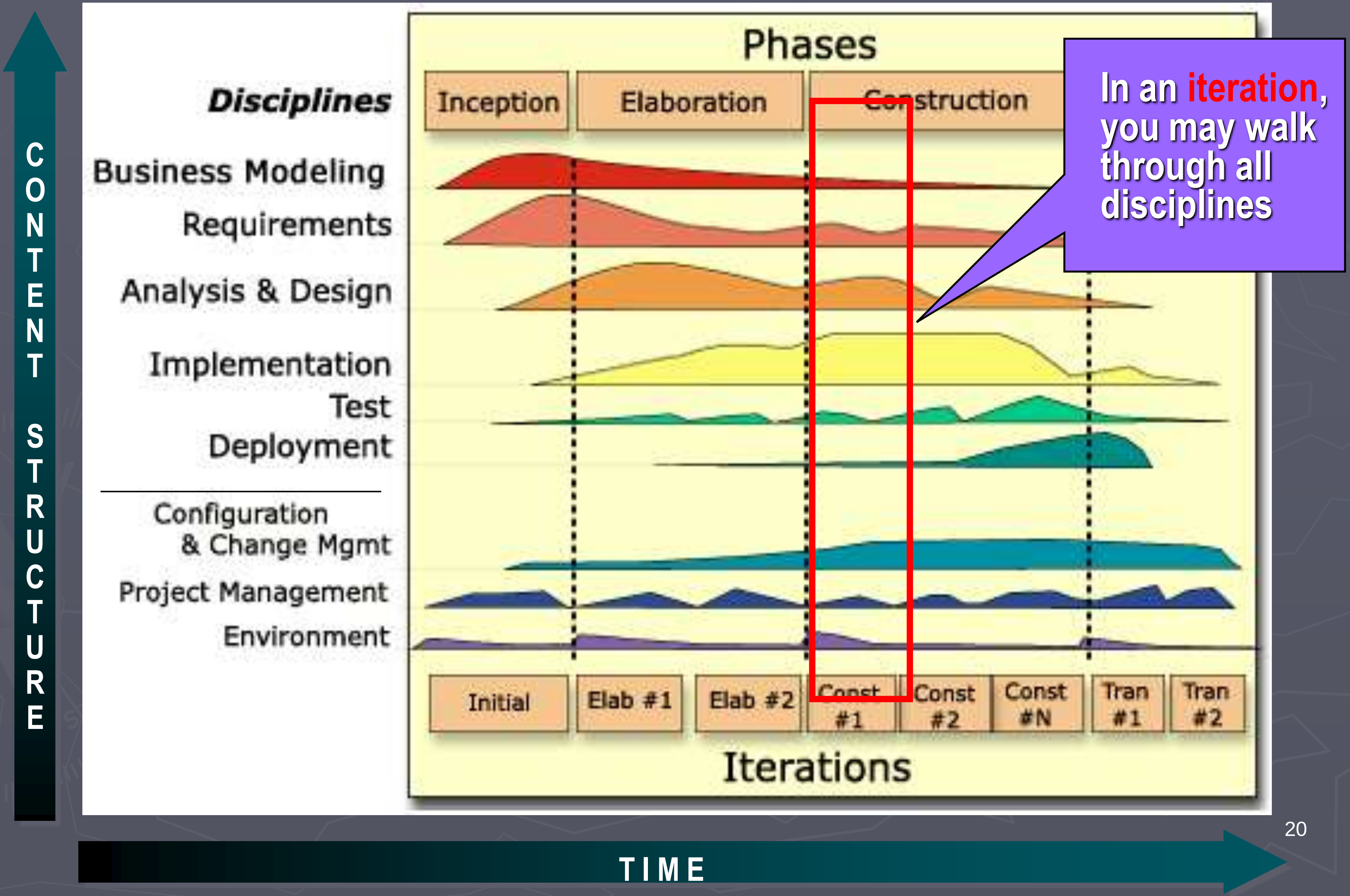


Iterative Development Characteristics

- **Critical risks are resolved before making large investments**
- **Initial iterations enable early user feedback**
  - Easy to resolve problems early.
  - Encourages user feedback in meaningful ways
- **Testing and integration are continuous** – assures successful integration (parts all fit)
  - Continuous testing.
- Objective milestones provide short-term focus
- Progress measured by assessing <u>implementations</u>
- **Partial implementations can be deployed**
  - **Waterfall method – no delivery**
  - **Incremental development? May be some great values in delivering key parts of application. Critical components delivered first?**
- **No big-bang approach!**

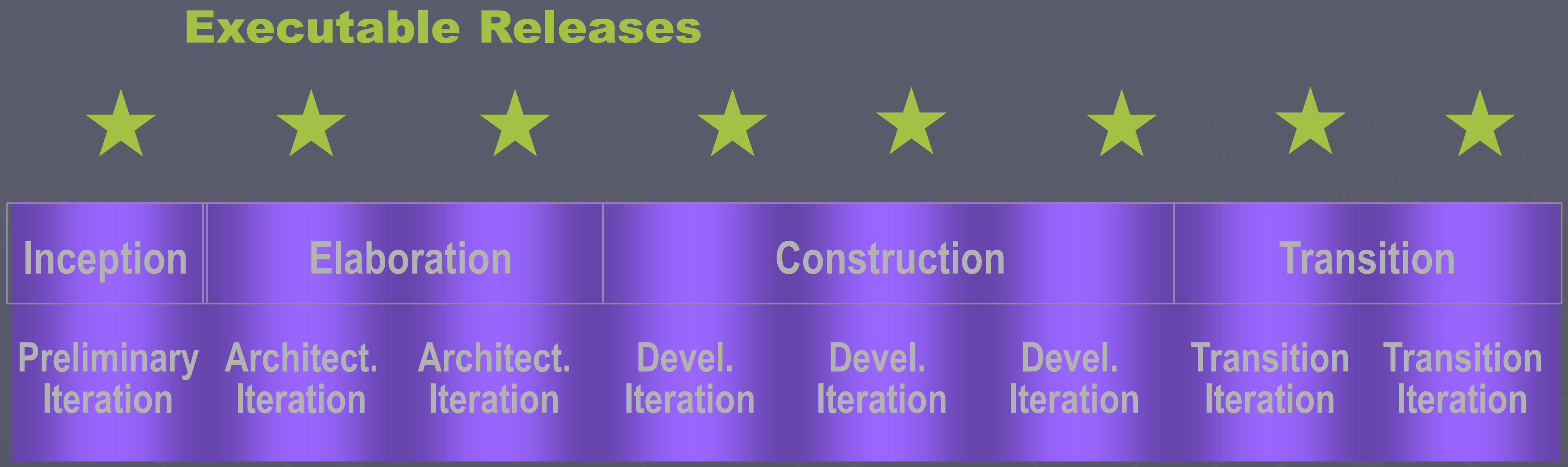
UP Lifecycle Graph – Showing Iterations

STUDY THIS!!!



In an iteration, you may walk through all disciplines

Unified Process Iterations and Phases



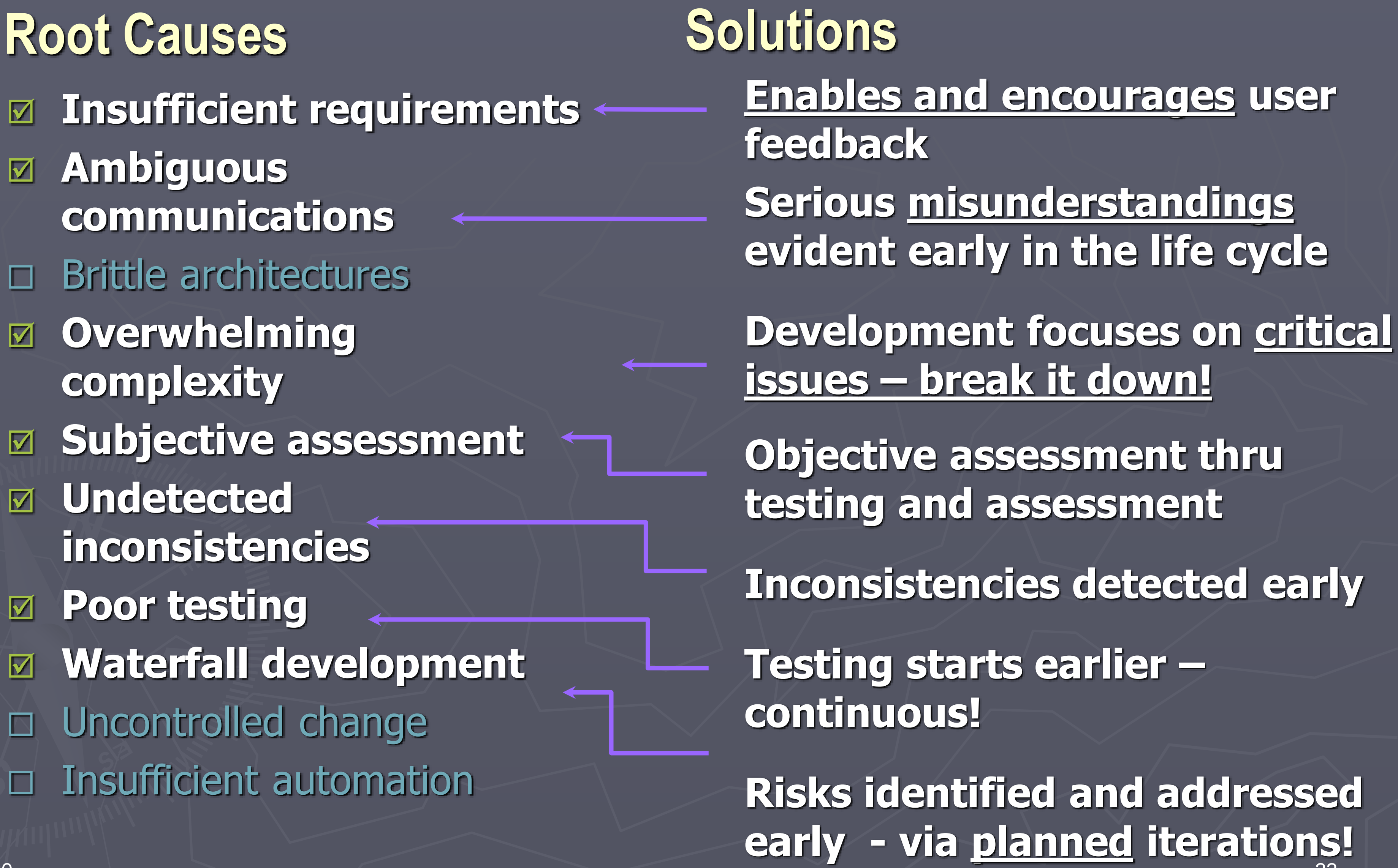
Executable Releases

Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architect. Iteration	Architect. Iteration	Devel. Iteration	Devel. Iteration	Devel. Iteration	Transition Iteration	Transition Iteration

An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an ‘executable release.’

(There is a lot of very important ‘key’ terminology used here… (cycle, iteration, phase, milestones, core disciplines, content of iterations, etc.…)

Problems Addressed by Iterative Development



Root Causes

- ☑ Insufficient requirements
- ☑ Ambiguous communications
- □ Brittle architectures
- ☑ Overwhelming complexity
- ☑ Subjective assessment
- ☑ Undetected inconsistencies
- ☑ Poor testing
- ☑ Waterfall development
- □ Uncontrolled change
- □ Insufficient automation

Solutions

Enables and encourages user feedback

Serious misunderstandings evident early in the life cycle

Development focuses on critical issues – break it down!

Objective assessment thru testing and assessment

Inconsistencies detected early

Testing starts earlier – continuous!

Risks identified and addressed early - via planned iterations!

No Free Lunch - Traps Abound…

- Major impacts on Project Managers, though….

- Trap: When the initial risks are mitigated, new ones emerge
  Do not do just the easy stuff, to look good.
  Keep re-planning based on all new information.
- Trap: Remember 'some' Rework enables you to enhance your solution
  Accommodate change early in the project
- Trap: Iterative development does **not** mean never to commit to a solution

- Monitor 'scrap and rework'
- Trap: Must Control "requirement creep, " however… Some
  clients will now naturally recognize many 'musts…'

Many Traps in Iterative Development

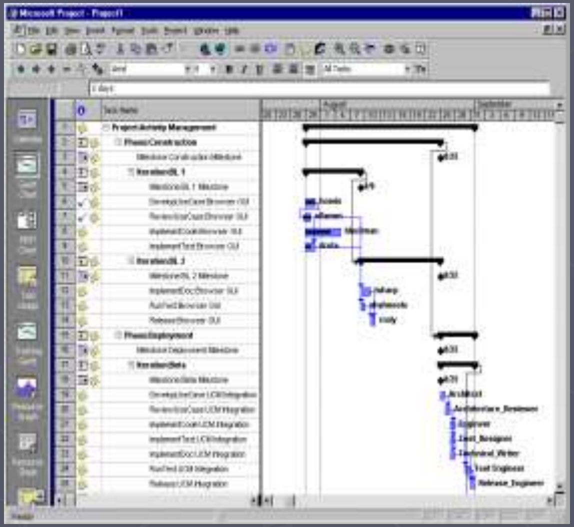
Here is another trap: Too long initial iteration

- Winning is fun. Winning teams work better than loosing teams
- **Better** to have a short initial iteration, than one too long
  - Cut scope if necessary (much more later)
- Avoid ‘analysis-paralysis’ by **time-boxing;** you can enhance in later iterations (more later)
- Establish an **even rhythm** for project (at least w/i a phase)
- Focus on results and deliverables, not activities

Iterations Are Time-boxed

- Work is undertaken within an iteration.
- The iteration plan **defines** the artifacts to be delivered, roles and activities.
- An iteration is clearly **measurable**.
- Iterations are **risk-driven**
- Iterations are **planned**.
- Iterations are **assessed**!
- Generally, initial iterations (in Construction) based on high risk and core functionalities!

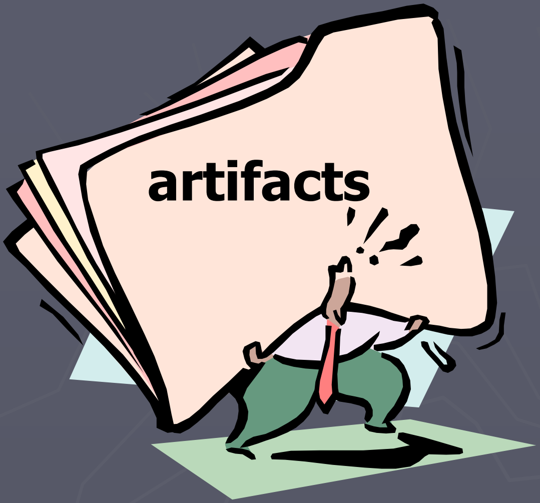
The Iteration Plan Defines….



Microsoft Project - Project1

	Task Name
1	Project Activity Management
2	Phase:Construction
3	Milestone Construction Milestone
4	Iteration:BL 1
5	Milestone:BL 1 Milestone
6	DevelopUseCase:Browser GUI
7	ReviewUseCase:Browser GUI
8	ImplementCode:Browser GUI
9	ImplementTest:Browser GUI
10	Iteration:BL 2
11	Milestone:BL 2 Milestone
12	ImplementDoc:Browser GUI
13	RunTest:Browser GUI
14	Release:Browser GUI
15	Phase:Deployment
16	Milestone Deployment Milestone
17	Iteration:Beta
18	Milestone:Beta Milestone
19	DevelopUseCase:UCM Integration
20	ReviewUseCase:UCM Integration
21	ImplementCode:UCM Integration
22	ImplementTest:UCM Integration
23	ImplementDoc:UCM Integration
24	RunTest:UCM Integration
25	Release:UCM Integration

The deliverables for that iteration.



The to do list for the team members

Problem: Fixed Plans Produced Upfront – Not Real Practical!

- Yet, senior management wants firm, fixed plans!
  - Part of their culture / upbringing/ experience
  - Necessary for 'planning' budgeting, etc. of resources, projects…. BUT:

- **Trap**: Fine-grained planning from start to end?
  - Takes too much time
  - Frustrating as change occurs (and it **will)**, if plans too fine-grained.

- **Know that:** Projects typically have some degree of uncertainty
- This makes detailed plans for the entire project meaningless

- **Does not mean that we should not plan**

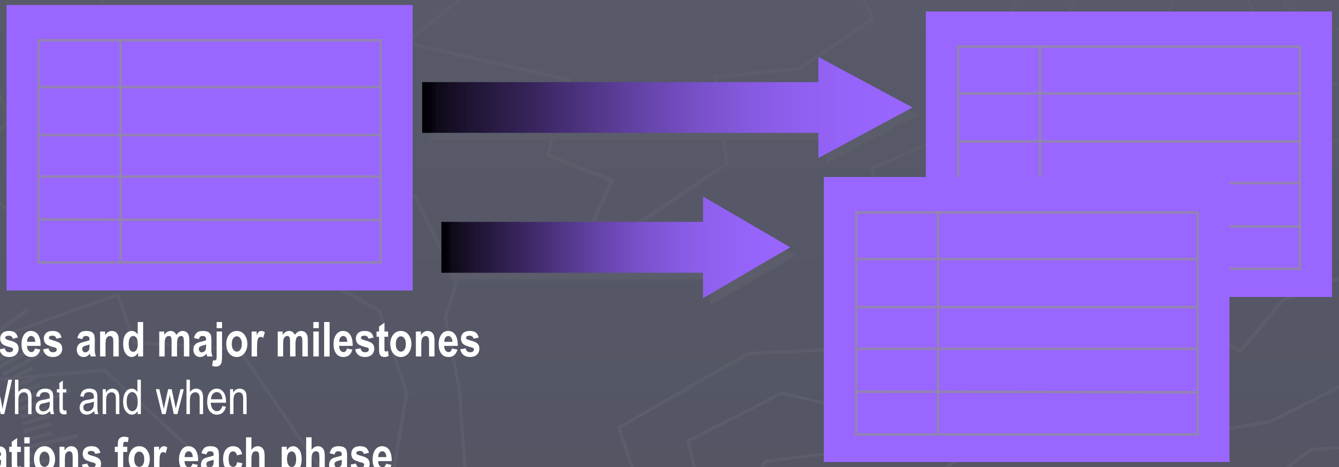
Solution:
Plan With Evolving Levels of Detail

<u>Coarse-grained</u> Plan:
Software Development Plan

<u>Fine-grained</u> Plans:
Iteration Plans

One For Entire Project

Next Iteration



Current Iteration

Phases and major milestones

- What and when

Iterations for each phase

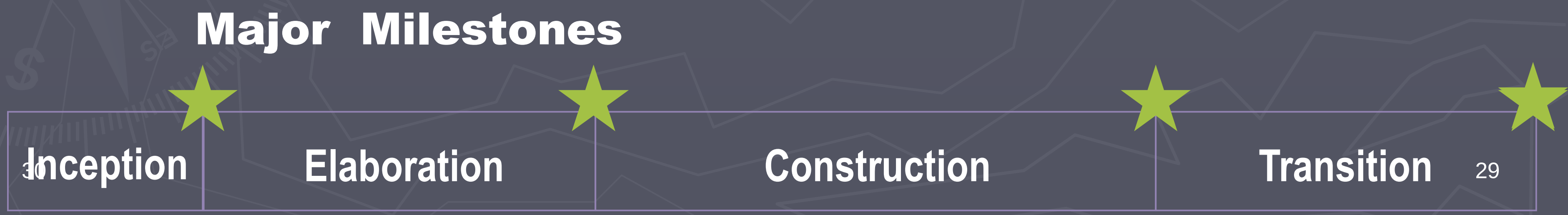
- Number of iterations
- Objectives and Duration

- Iterative Development <u>does not mean less work</u> and shorter schedule
- It is about greater <u>**predictability**</u>

Progress is made against MILESTONES

- In the Unified Process:
  - Each phase is defined by a milestone.
  - Progress is made by passing milestones.
  - Milestones measure success

- Phases - NOT TIMEBOXED.
- Iterations ARE TIMEBOXED.



Major Milestones

Inception

Elaboration

Construction

Transition

Summary

- Much more about iteration and iteration planning later in the course…

- You will see some of these again – and, more importantly, use this information in your own iteration planning.