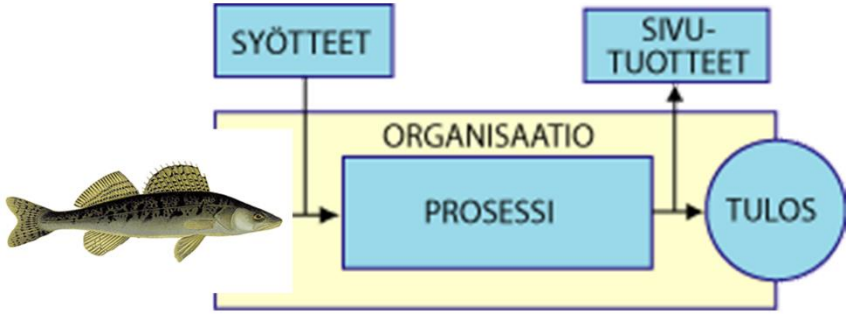
SOFTWARE ENGINEERING MODELS AND MODELING

ANTTI KNUTAS (D.SC.)

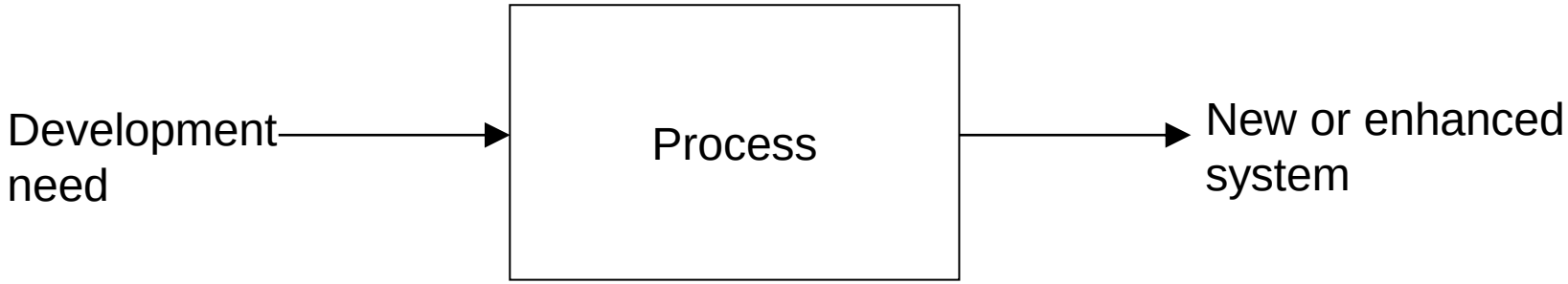
Open your mind. LUT.
Lappeenranta University of Technology

SOFTWARE PROCESS MODELS

LECTURE 2



PROCESSES AND PROCESS MODELS



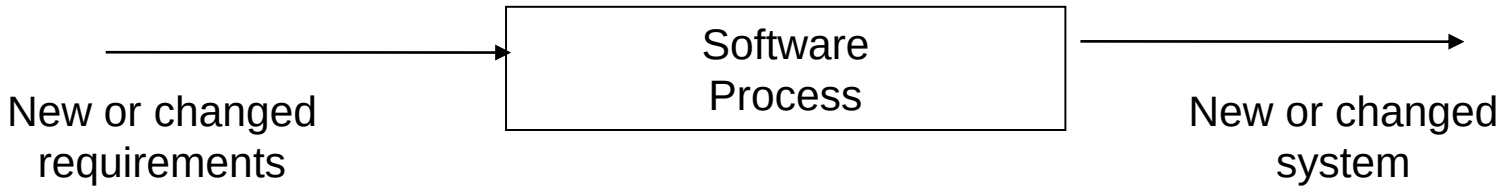
Objective: by using a defined process, system development becomes predictable and controllable

Process model

- A definition of a standardized development process
- Can be made in-house – commercial ones exist as well:
  - Rational Unified Process
  - Microsoft Solutions Framework
- Defines all necessary aspects and phases of the development process

What is a process?

- Process defines
  - **who** does,
  - **what** does,
  - **when** does and
  - **how** does.



TYPES OF PROCESS/LIFECYCLE MODELS

Lifecycle model

- Defines the main phases of a development project and the pattern how the phases link to each other

Most common high-level process (lifecycle) models:

- Waterfall model
- More adaptive models
  - Spiral/iterative model
  - Incremental model
  - Prototyping
  - Radical light-weight models (e.g. XP, Agile methods)

WATERFALL DEVELOPMENT MODEL

The phases are sequential

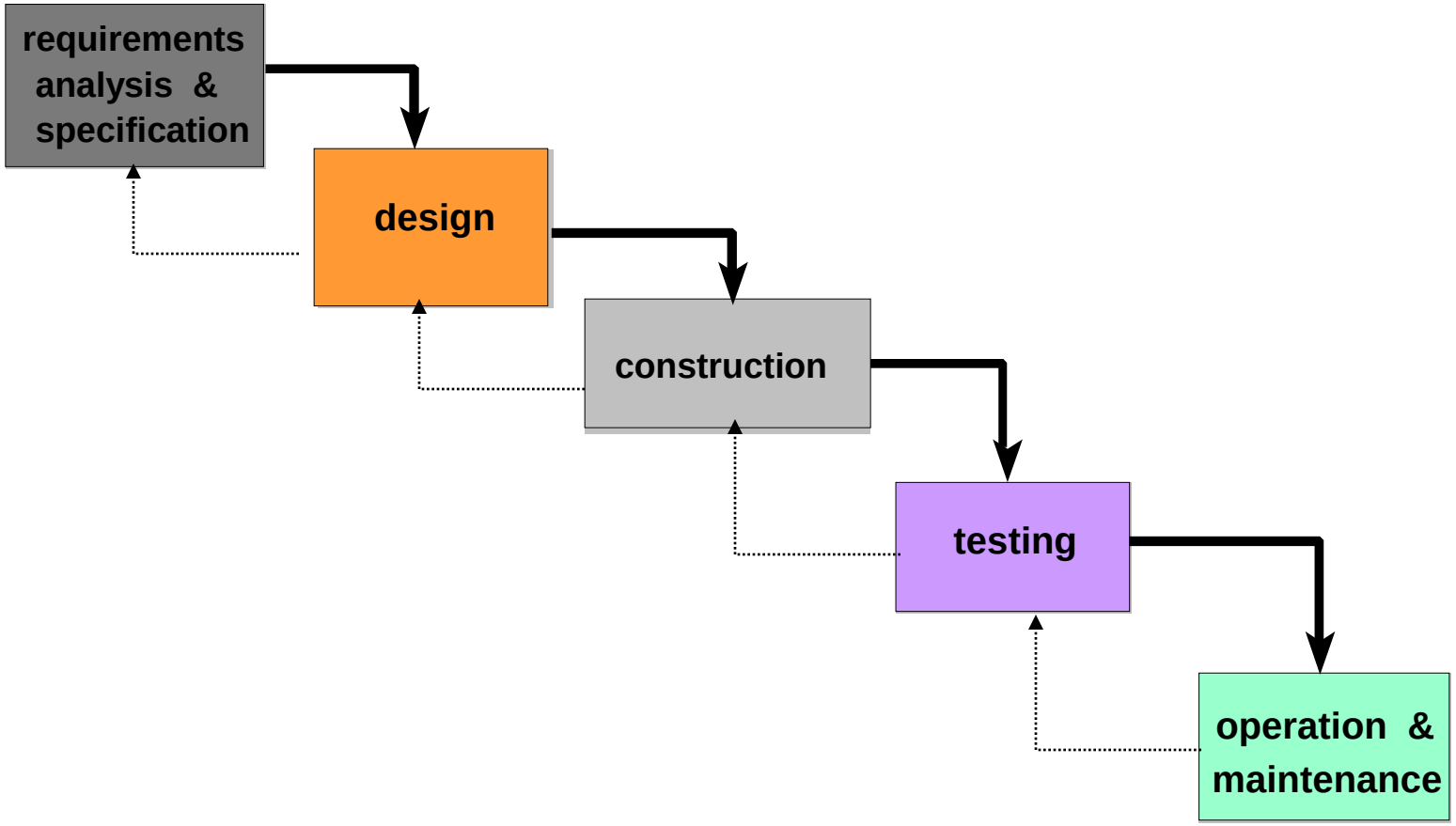
A phase delivers documents to the next phase

Most of milestones, documents, reviews, and decision points are located at the end of each phase

Pure waterfall – no overlap of phases

“feedback loops” are often introduced to allow return to previous phases (e.g., design analysis leads back to modification of the requirements)

ONE EXAMPLE OF A WATERFALL



WHAT IS WRONG WITH WATERFALLS?

Waterfall model is built-in to every engineer's thinking

- Specify → design → construct → test → take into use

Commercial contracts are usually defined using waterfall model as the basis

- Payments are located at the ends of phases and require defined deliverables

Project managers usually use waterfall model as the basis of planning

Why is there a growing criticism against waterfall model in the professional software development community?

WHAT IS WRONG WITH WATERFALLS?

Waterfall model is universally known, simple and powerful – yet it introduces many tough challenges to software development processes.

- There is no way to collect all requirements at the start.
- There is no way to change requirements.
- It expects that people are never wrong.
- It expects that platform will not change.
- It expects that platform is known very well before implementation.
- It expects that problems and bugs are easy to locate and fix from the final product :: the product is always correctly designed and built, save for some bugs.

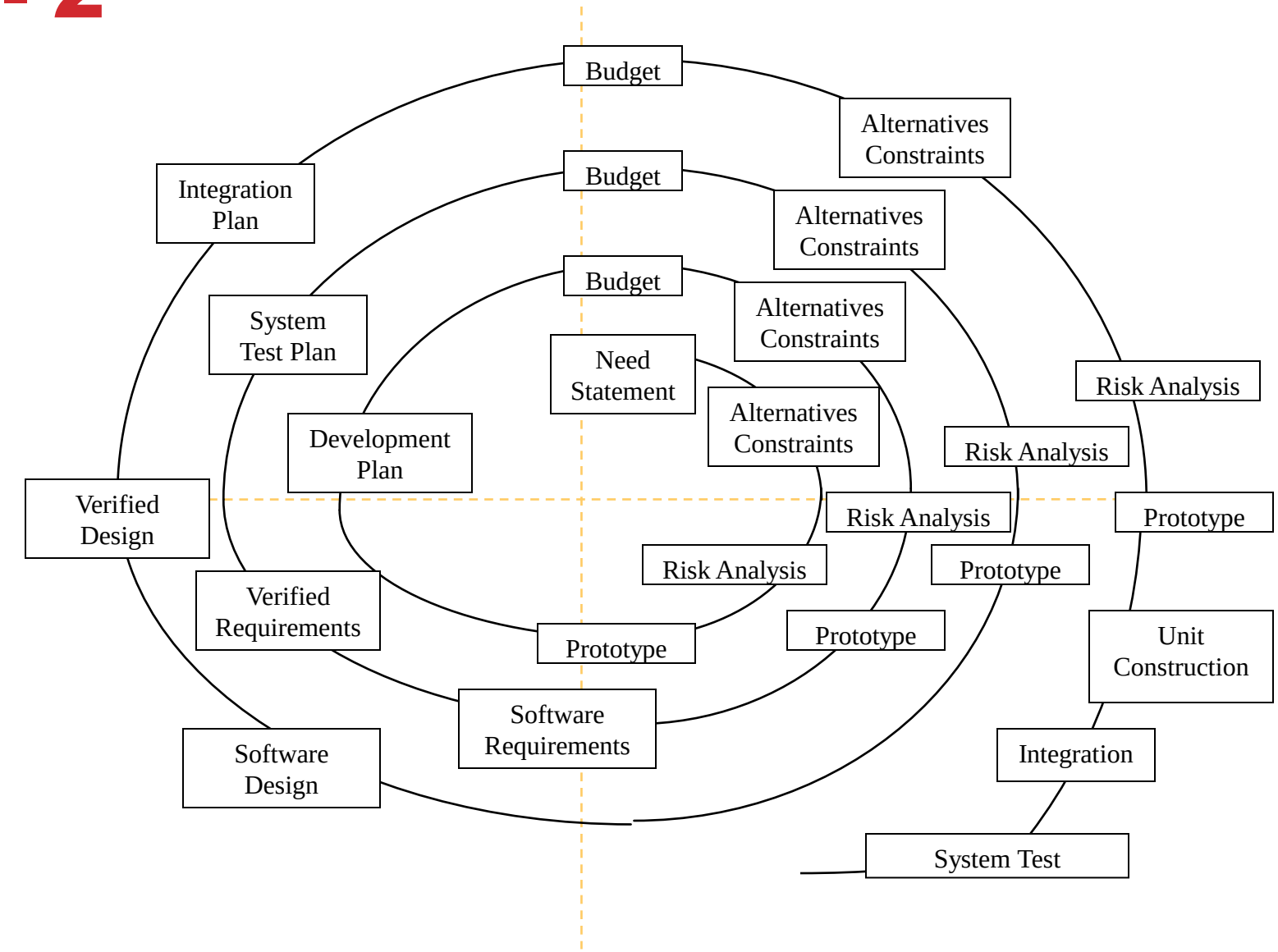
A RESPONSE TO THE PROBLEMS OF THE WATERFALL – SPIRAL MODEL

The Spiral Lifecycle combines elements of the waterfall lifecycle model, along with an emphasis on the use of risk management techniques.

Each cycle includes the following phases:

- Determine goals, alternatives, constraints
- risk analysis
- prototype development
- product development and verification
- planning for next phase

SPIRAL DEVELOPMENT MODEL - 2



ITERATIVE MODEL

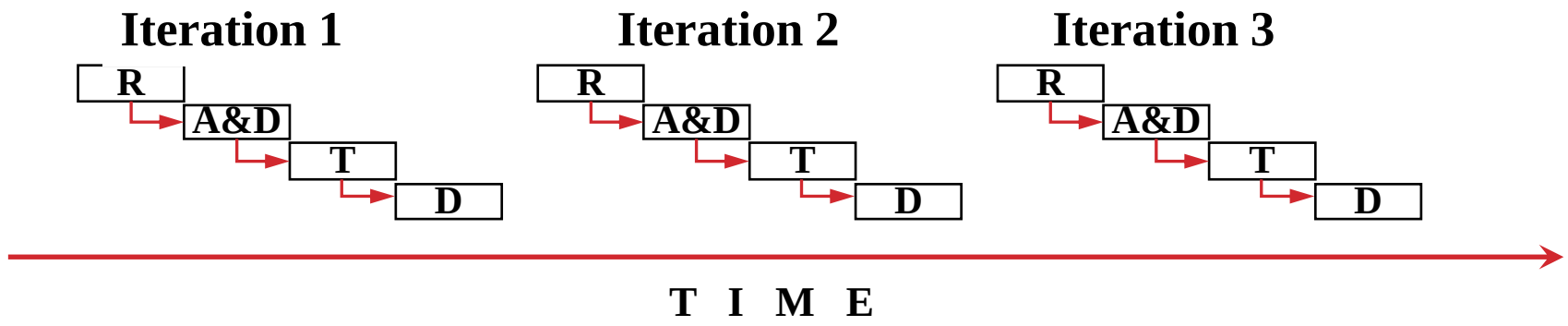
Another version of the spiral model

E.g. Rational Unified Process (RUP)

After initial planning, the project consists of individual iterations that each have a full development lifecycle

After the planned iterations the system can be deployed

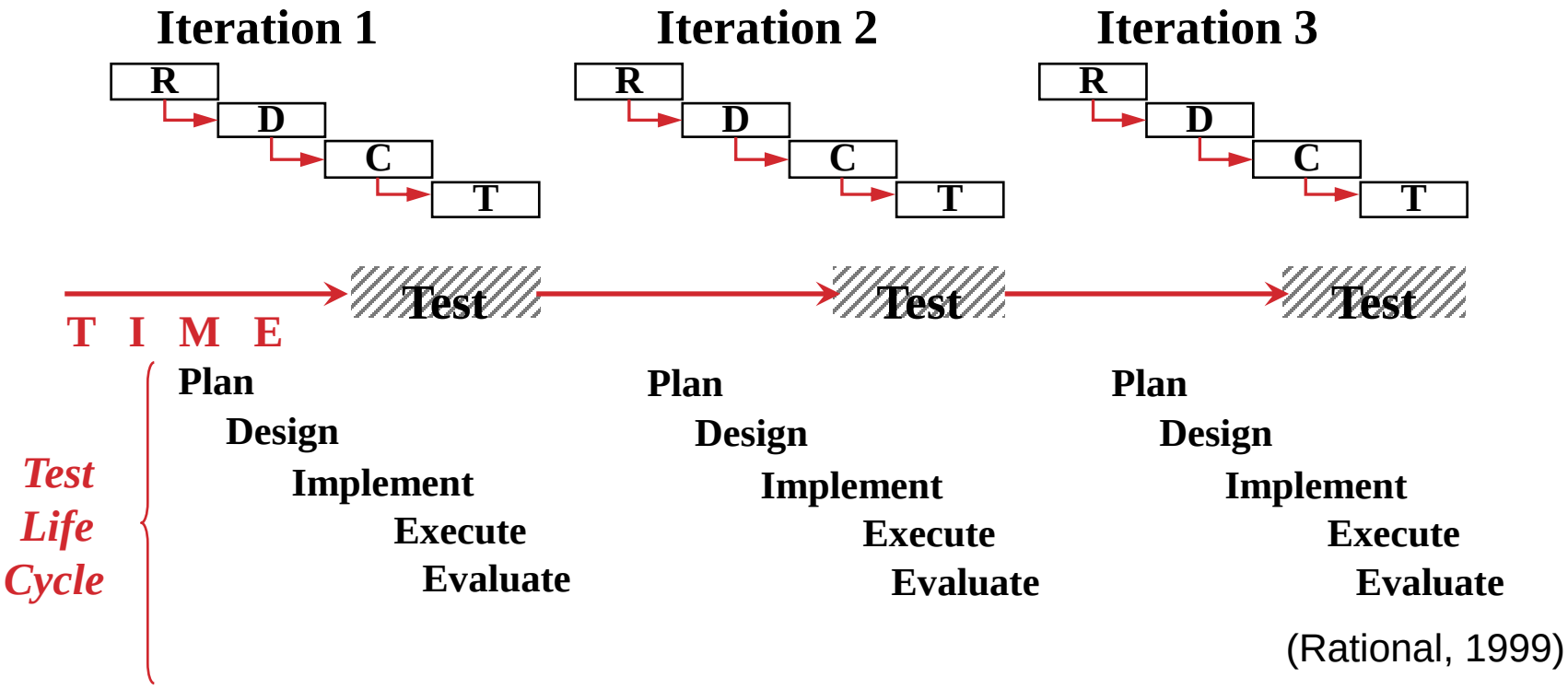
ITERATIVE DEVELOPMENT



A project consists of a series of ”mini projects”

- Each iteration produces executable and testable result
- Iterations are made in risk order
  - The iteration with the greatest risk will be done first
  - The objective is to decrease risks as fast as possible
  - Enables user feedback
- Requires skilled and experienced project management

TESTING AND ITERATIVE DEVELOPMENT



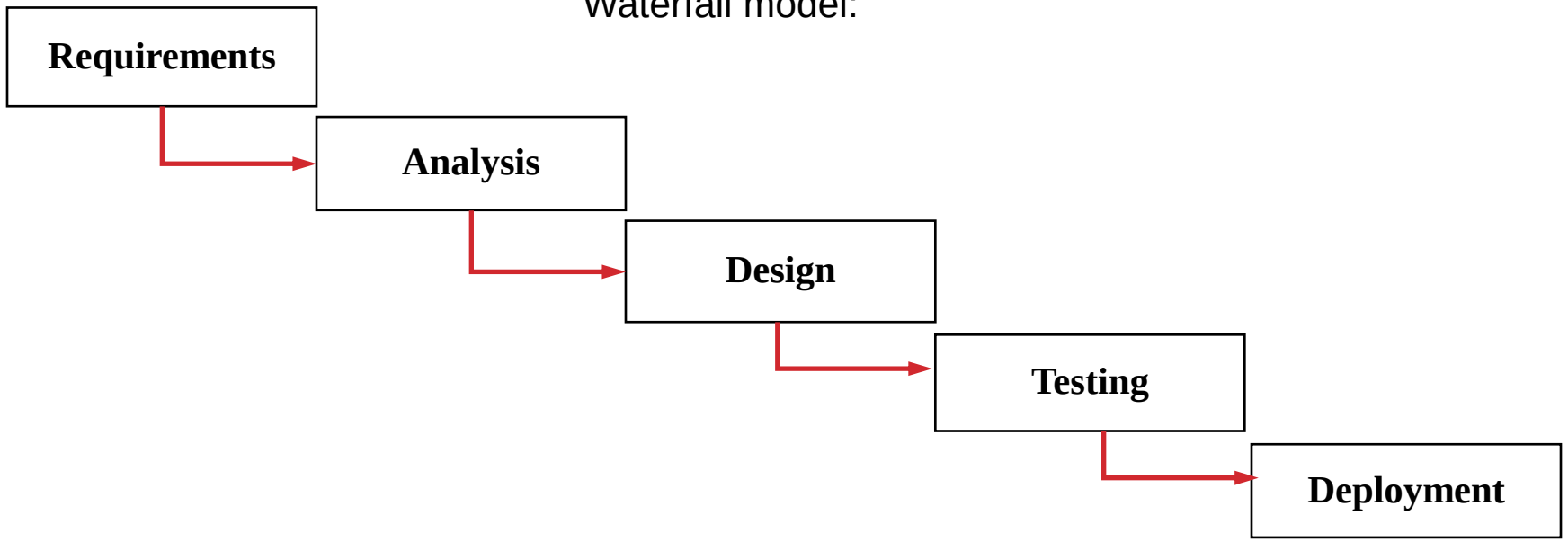
(Rational, 1999)

The goal is to find faults as early as possible

- Significant cost effects

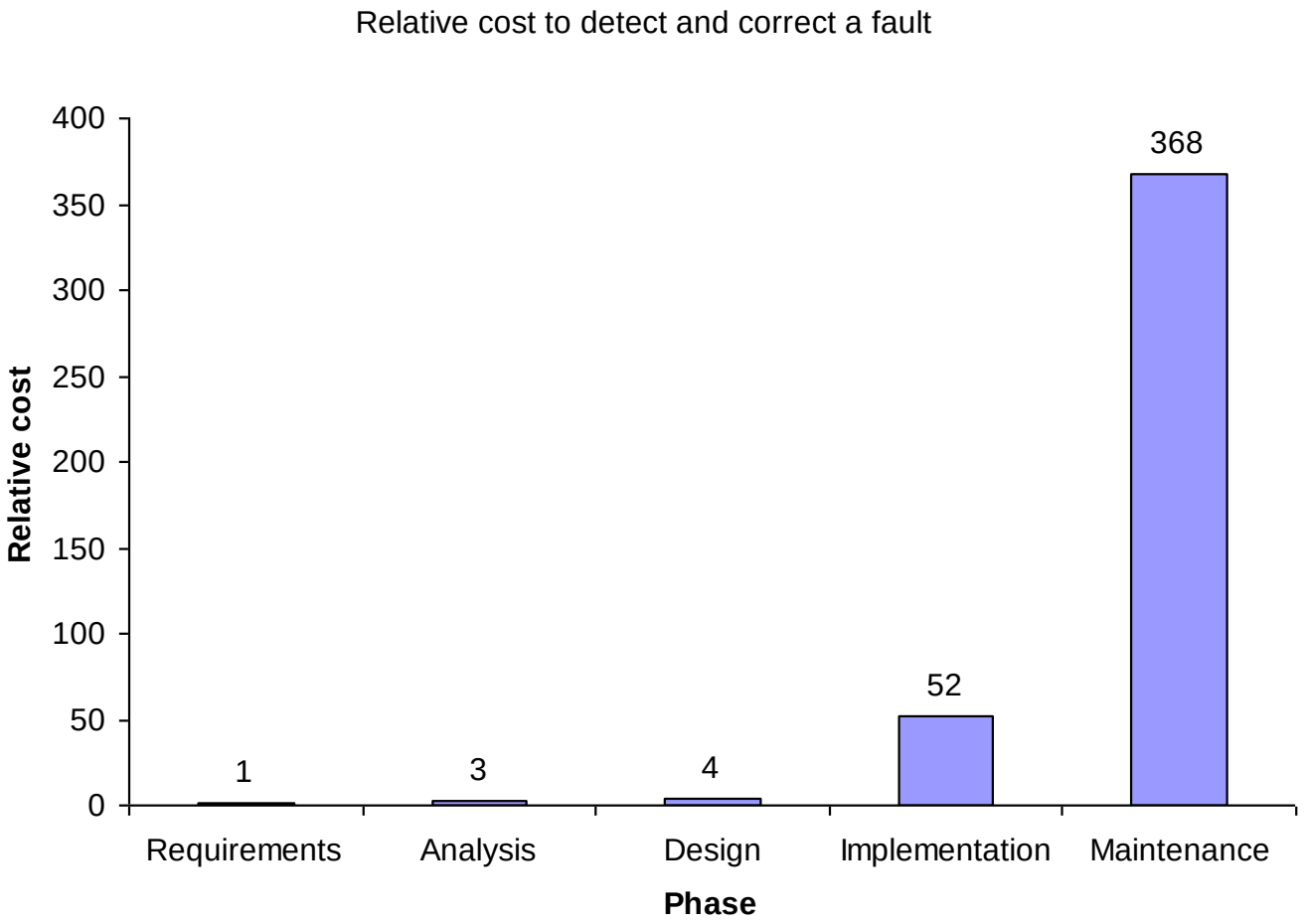
WATERFALL MODEL VS. ITERATIVE DEVELOPMENT

Waterfall model:



- When applied as is, the waterfall model leads to late recognition of risks
  - A defect in requirements specification identified in testing or deployment may prove very expensive

RELATIVE COST OF A FAULT (SCHACH, 2002)



INCREMENTAL DEVELOPMENT

A version of the spiral/iterative model

First, an overall architecture of the total system is developed

Then, the detailed increments and releases are planned

Each increment has its own complete lifecycle.

The increments may be built serially or in parallel depending on the nature of the dependencies among releases and on availability of resources.

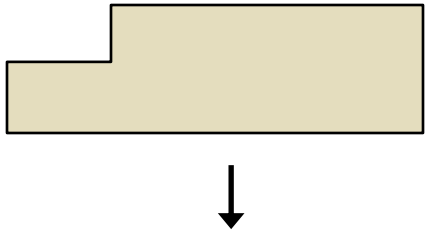
Each increment adds additional or improved functionality to the system.

The implementation of increments may end e.g.

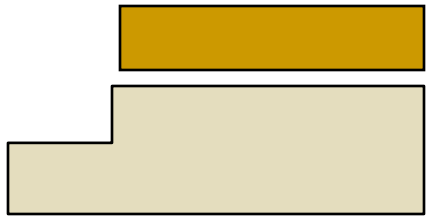
- When the money runs out (budget-based development)
- When the deadline arrives (time-to-market)
- When requirements are met

AN EXAMPLE OF INCREMENTS

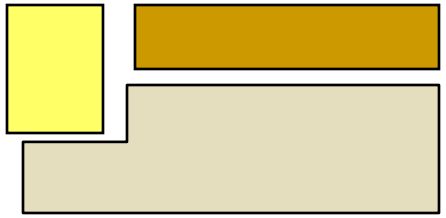
1st increment



2nd increment



3rd increment



OTHER APPROACHES AND RELATED AREAS

Radical development approaches

- Prototyping
- RAD – Rapid Application Development
- Agile approaches (more on them later)
- Participative development
- Soft Systems Methodology

Related areas

- Software Process Improvement, CMM
- Open source development
- Development of Enterprise Resource Planning (ERP) systems

PROTOTYPING

When a complete and detailed understanding of the requirements cannot be achieved it could be useful to demonstrate the system with a prototype

- A working model of (parts of) a system, which emphasizes specific aspects
- Exhibits the essential features of the system
- May evolve into the actual production system or may be used only for experimentation

As a development approach prototyping has

- High degree of iteration
- Very high degree of user participation
- Extensive use of prototypes

BENEFITS AND WEAKNESSES OF PROTOTYPING (FITZGERALD & AL. 2002)

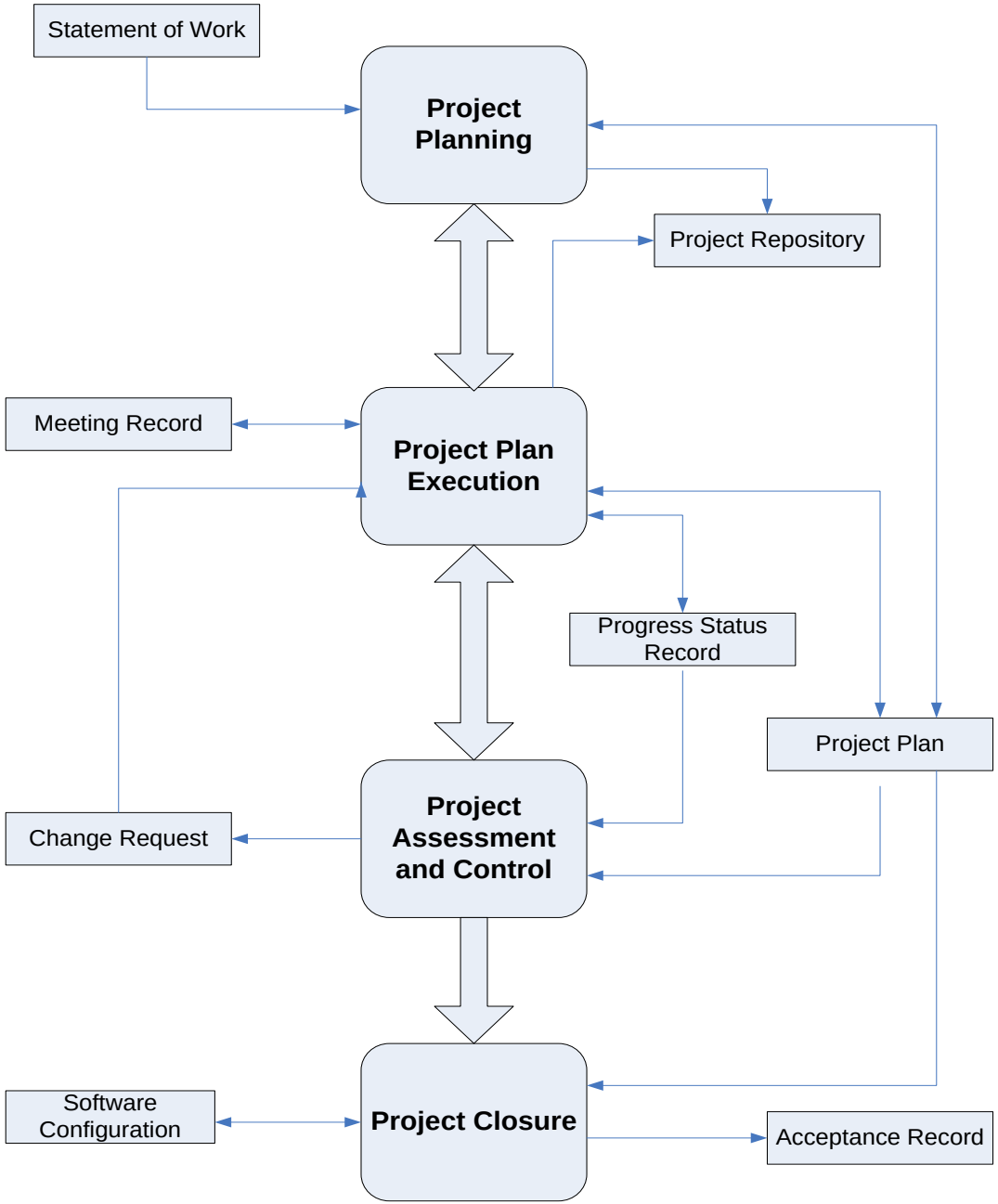
Benefits

- Facilitates the specification of requirements with greater precision
- Enforces user participation
- Hands-on experience enables better learning process for both developers and users
- Requirements are understood – quality and user satisfaction
- System appears to be developed more quickly

Weaknesses

- Project management is more complex
- Prototyping requires special tools
- Systems integration may fail
- Alternatives and business requirements may be studied poorly
- May create unrealistic expectations about schedules
- Is often impossible in large projects
- Requires significant user commitment

ISO/IEC 29110 FOR SMALL ORGANIZATIONS



OPEN SOURCE DEVELOPMENT

Characteristics

- Requirements come from open community of users and developers
- Developers choose what they want to work on
- Software architecture is designed modularly
- The choice of development tools is limited and often come from other open source projects
- Software process is informal and a small group of technical experts decide about the releases
- The project often runs on personal incentives instead of profits

➔ The development process differs much from the process of commercial development. The lessons may not be interchangeable (?)

SHORT ANSWER

Discuss: What kind of development processes have you been involved in? What type it was?

AGILE APPROACHES

AGILE APPROACHES

Agile manifesto (2001)

Many published methods

- XP
- SCRUM
- Adaptive software development …
- Continuous methods

COWBOY CODING

A contrast to using methods - software development without any method

- Developers do what they feel right
- Software development is a heroic effort made by artists/warriors

Cowboy coding may work in small tasks, but fails in any larger effort

AGILE VIEWPOINT: A DIFFERENT APPROACH

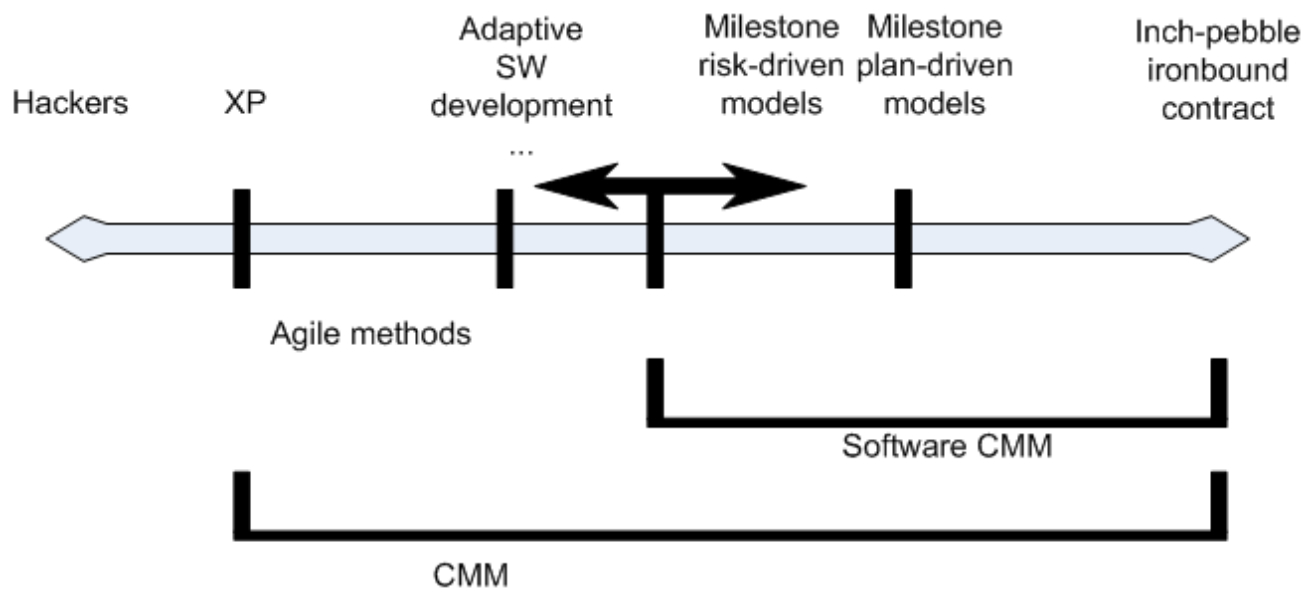
In agile development methodologies the documented process and practices are not emphasized

Individuals are respected and *the team* can decide their own practices

Quality management on team or project level is emphasized

The Methodology-Growing Technique is one example of agile approach to quality assurance

IDEA: PROCESS MODEL ACCORDING TO THE ACTUAL NEED



Not necessarily this straightforward, but the general idea is in “preparing for thing no matter how improbable” vs. “adapting to the problems that arise”.

METHODOLOGY-GROWING TECHNIQUE

- "On-the-fly methodology construction and tuning"
  - Choosing and documenting the practices
  - Evaluating the practices
  - Identifying *non-compliance* issues and *bad practice issues*
  - Finding resolutions and tuning the process
- Focusing into the team
  - What was done and how well it worked out
  - And how to become better
  - Instead of documents, independent evaluation, and reporting issues

A. Cockburn, Agile Software Development, 2002

METHODOLOGY GROWING TECHNIQUE

Discover the strengths and weaknesses of your organization through short project interviews

1. **Create a base methodology at the start of the project**

- **Adjust it together with the development team**

2. **Run a small interview in the middle of an iteration**

- **“Are we going to make it, working the way we are working?”**

3. **Hold a team reflection workshop after each iteration**

4. **(Post-project review)**

REFLECTION WORKSHOP
(TEAM RETROSPECTIVE)

A team meeting lasting typically around half a day

- **Evaluating the effects of each element of the process**
  - After delivering a running tested increment
  - Only then you can see what was underdone and what was overdone
- **Main questions**
  - “What did we learn?”
  - “What can we do better?”

- **Results can be**
  - Tightened standards
  - New practices and conventions
  - Changed practices
  - Dropping unnecessary things
  - Changes in team organisation
  - …

REFLECTION WORKSHOP - DELIVERABLE

Worked well

describes the current practices of the team

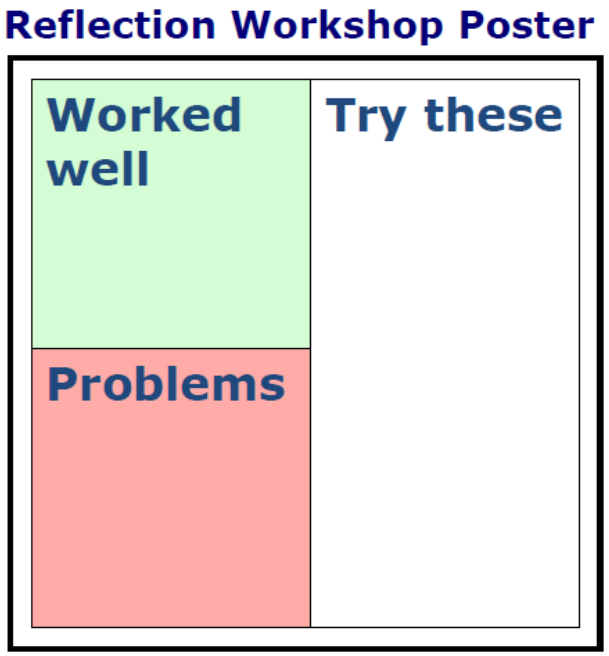
Problems

Lists the identified issues

Try these

presents the agreed resolutions for the issues and other improvement actions

- Evaluation criteria is not documented



COMPARISON OF THE METHODS; ”THE CASTLE VS. THE TIGER”

CMMi, ”plan-driven practices”	**Method**	Agile practices
Independent QA team	**Organization**	Integrated into the project team
Compliance to documented processes	**Ensuring**	Applicability and improvement of the current practices and processes
Against predefined criteria	**Evaluation Criteria**	Identifying issues and problems
Documents & processes & control	**Focus**	Productivity & quality & customer
Formal; reporting to management	**Communication**	Informal; supporting the team
Prepare against all possible issues	**General Strategy**	React quickly to issues that arise
Outside, observing process	**Viewpoint**	Inside, active participant
Amount of documented data	**General Issue**	Availability of data

REFLECTIVE PRACTICES

Don’t repeat same mistakes – learn to avoid them!

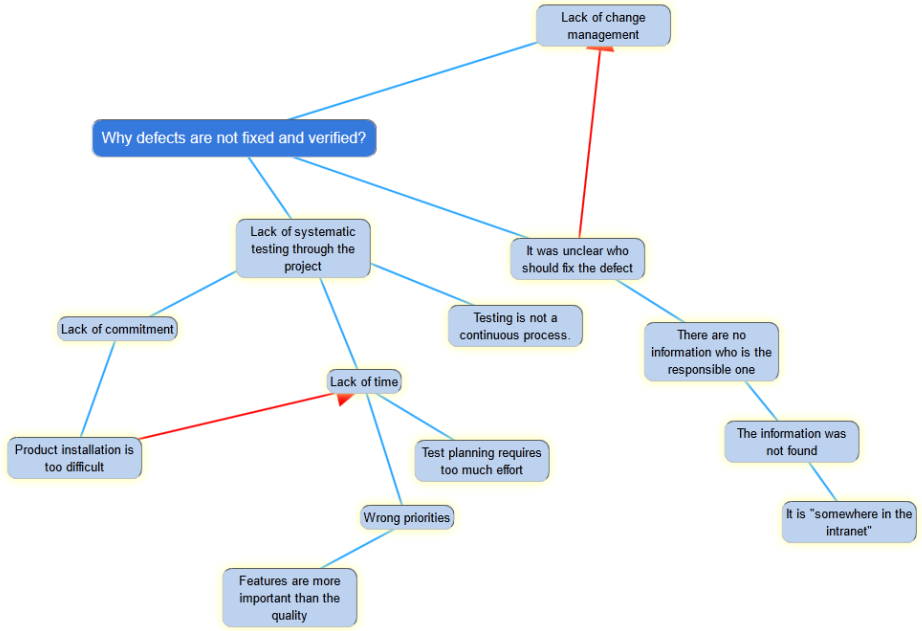
”Learn by doing (mistakes)”

Organizational learning (continuous improvement)

- **Project reflections (e.g. post-project review, retrospective, postmortem, defect causal analysis)**
  - What went well and what went wrong in the project?
  - What succeeded and what was challenging in the last sprint?
- **Knowledge management**
  - Capture and refine the lessons learned
  - Share the lessons learned over the organization

ROOT CAUSE ANALYSIS

- **The key for effective problem prevention is to know why the problem occurs (Rooney, 2004) (Kalinowski, 2008)**
- **Root Cause Analysis (RCA) is a structured investigation of a problem to detect the problem causes that need to be eliminated (Latino and Latino, 2006)**



SOME PRACTICES OF AN AGILE DEVELOPER

- Integrate early integrate often
- Keep your project releasable at all times
- Automate acceptance testing
- Use automated unit tests
- Use it before you build it – TDD
- Write code in short edit/build/ test cycles
- Emphasize collective ownership of code
- Keep a solutions log
- …

- Keep it simple
- Don’t fall for the quick hack
- Write code to be clear not clever
- Warnings are really errors
- Provide useful error messages
- Be a mentor
- Share code only when it’s ready
- Review code
- Keep others informed of the status of your work
- …

V. Subramaniam & A. Hunt, “*Practices of an Agile Developer*”, The Pragmatic Bookshelf, 2006.

SHORT ANSWER

Discussion: What kind of good practises of an agile developer have you seen in prorgamming projects? How was it useful?

EXAMPLES OF AGILE MODELS

EXTREME PROGRAMMING

XP

- Creators Kent Beck, Ward Cunningham and Ron Jeffries
- Describes a set of day-to-day practices for planning, designing, coding and testing
- Enforces certain values

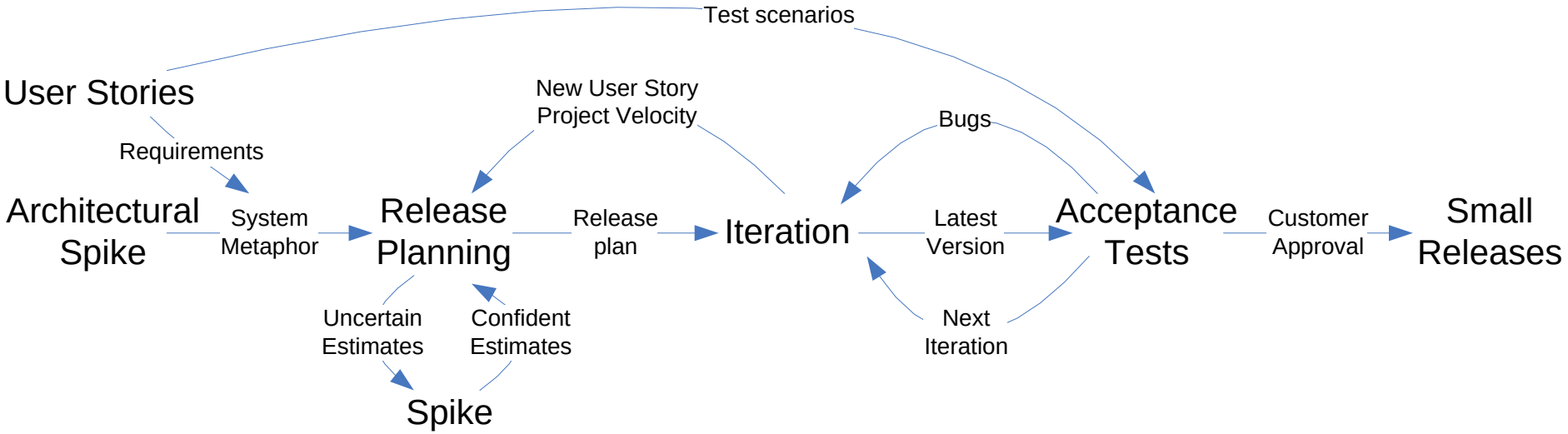
Objective is to reduce the cost of change

- Responsiveness to customer needs
- Aim at high quality

XP requirements

- Small teams (2-12 programmers)
- An extended team with customers
- Automated unit and functional tests
- Joint working environment (or at least easy knowledge sharing system)

EXTREME PROGRAMMING PROJECT (WELLS, 2006)



XP PRINCIPLES: PLANNING

Planning

- User stories are written.
- Release planning creates the schedule.
- Make frequent small releases.
- The Project Velocity is measured.
- The project is divided into iterations.
- Iteration planning starts each iteration.
- Move people around.
- A stand-up meeting starts each day.
- Fix XP when it breaks.

XP PRINCIPLES: DESIGNING

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC (Class-Responsibility-Collaboration) cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

XP PRINCIPLES: CODING

Coding

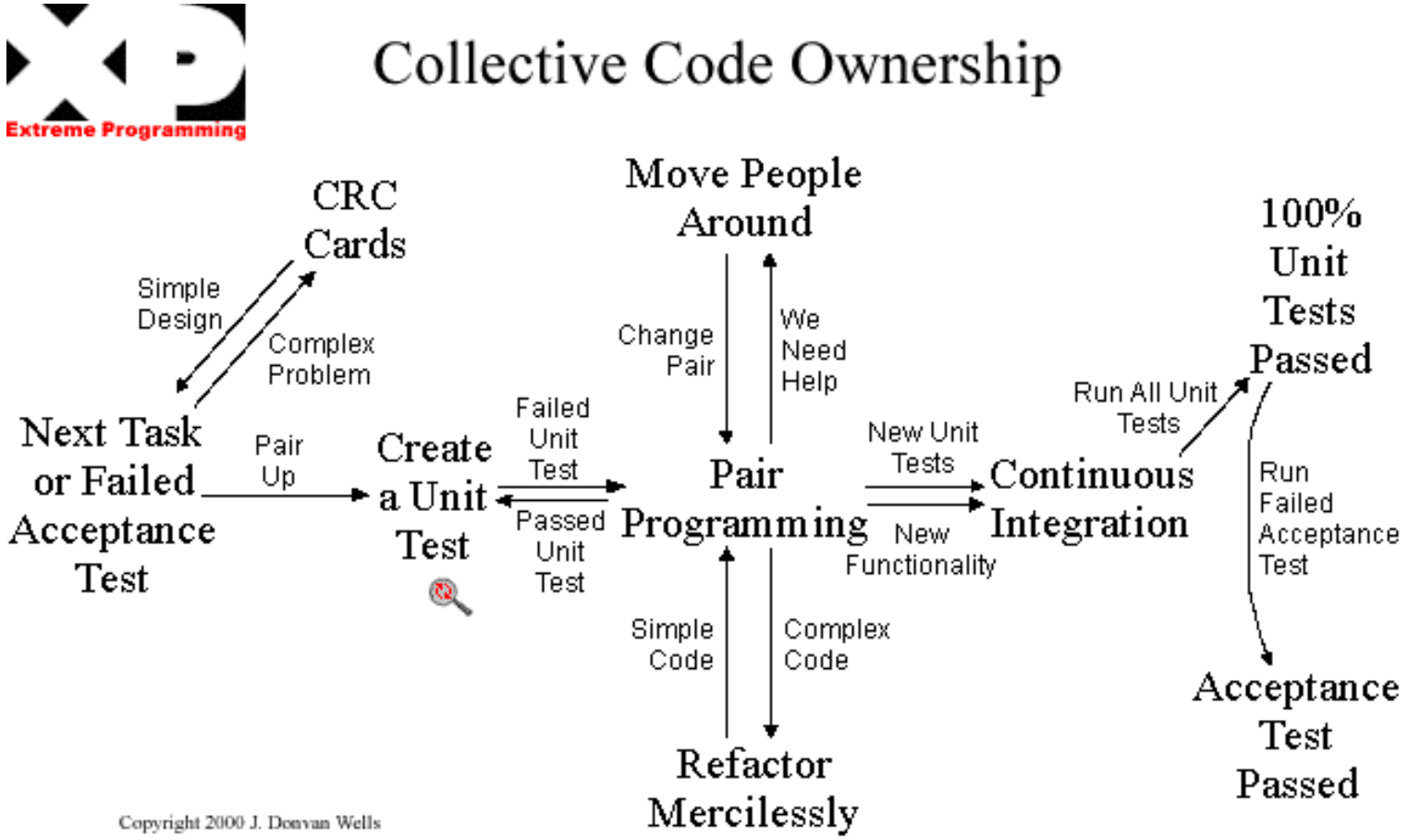
- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Use collective code ownership.
- Leave optimization till last.
- No overtime.

XP PRINCIPLES: TESTING

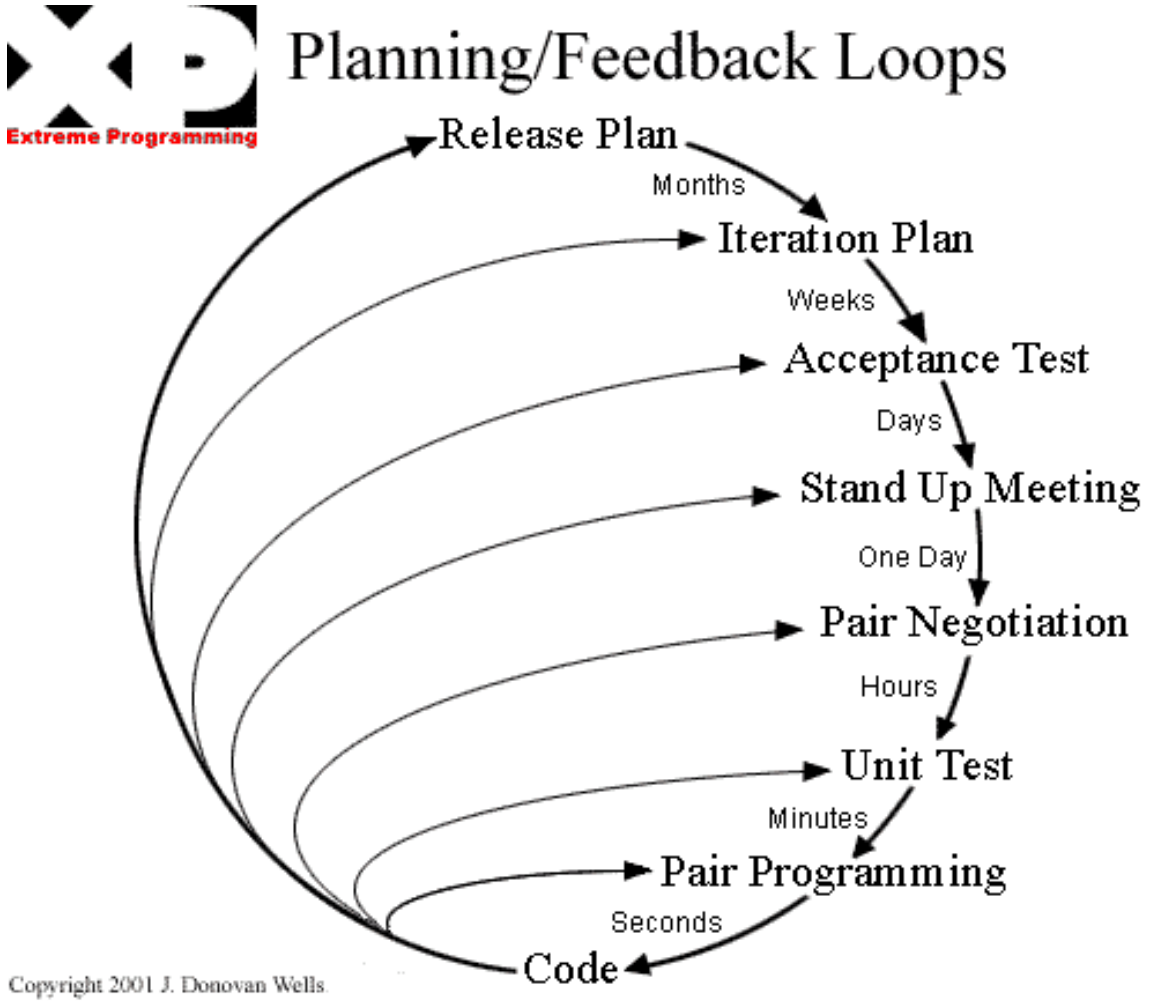
Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.
- Test driven development → tests written to break code; code is done when no-one comes up with a test that could cause errors

XP: COLLECTIVE CODE OWNERSHIP



XP: PLANNING/FEEDBACK LOOPS

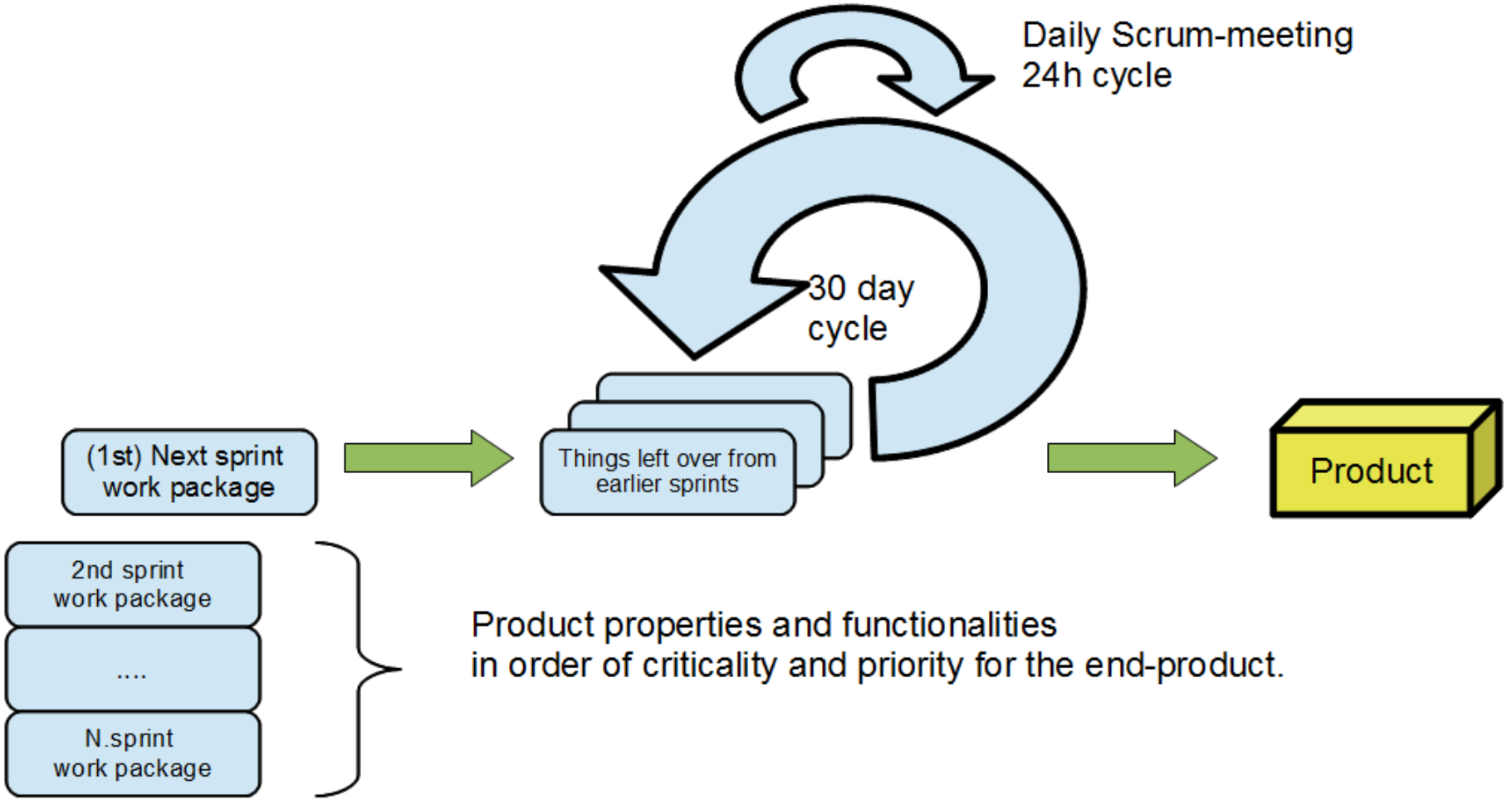


SCRUM: AN AGILE PROJECT FRAMEWORK

Scrum emphasizes more project management aspects than for example XP

- Sprints
  - 2-4 week iterations
- Task management artifacts
  - Product backlog, Sprint backlog, Burndown chart
- Developer roles
  - Product owner, Scrum master, Team
- Development ceremonies
  - Sprint planning meeting, Daily scrum meeting, Sprint review meeting

SCRUM MODEL


Daily Scrum-meeting
24h cycle
30 day
cycle
(1st) Next sprint
work package
Things left over from
earlier sprints
Product
2nd sprint
work package
....
N.sprint
work package
Product properties and functionalities
in order of criticality and priority for the end-product.

SCRUM: SPRINTS

A project is split into less than 30 day iterations called Sprints

One sprint realizes one set of new requirements, including the design, implementation and testing of the features.

The requirements which are not finished in time, will be included in the backlog, which will be sorted out usually at a separate sprint or at the beginning of the next sprint.

Sometimes also “QA sprints”; focus on just doing additional testing work, refactoring modules or in quality assurance tasks.

SCRUM: TASK MANAGEMENT ARTIFACTS

Product backlog

- A prioritized list of requirements, stories, features, etc.
- The things that the customer wants, described using the customer's terminology

Sprint backlog

- Features from the product backlog are split into items requiring less than ~16 hours of work
- Sprint planning selects the items and compares work estimates with the product backlog

Burndown chart

- Day-by-day cumulative work remaining in a Sprint

SCRUM: DEVELOPER ROLES

Product owner

- Defines the features
- Is responsible for the business aspects
- Adjusts features and priorities before every sprint

Scrum master

- Manages the project
- Organizes the meetings and ceremonies
- Controls the progress

Team

- 7 +/- 2 members
- Does what is needed within project boundaries to reach the Sprint goal

SCRUM: CEREMONIES

Sprint planning meeting

- The product owner develops a plan for a product or a project
- When enough of the Product backlog is defined, the first thirty-day Sprint is launched
- In Sprint planning meeting, a detailed plan for the iteration is developed

Daily scrum meeting

- Each day the Scrum master leads the 15-minute Daily scrum meeting
- Each team member answers to the questions: what did I do yesterday, what did I do today, and what impediments got in my way?

Sprint review meeting

- Happens at the end of a Sprint
- Includes demonstration, review, next sprint prioritization and goal-setting, and a retrospective of the Sprint

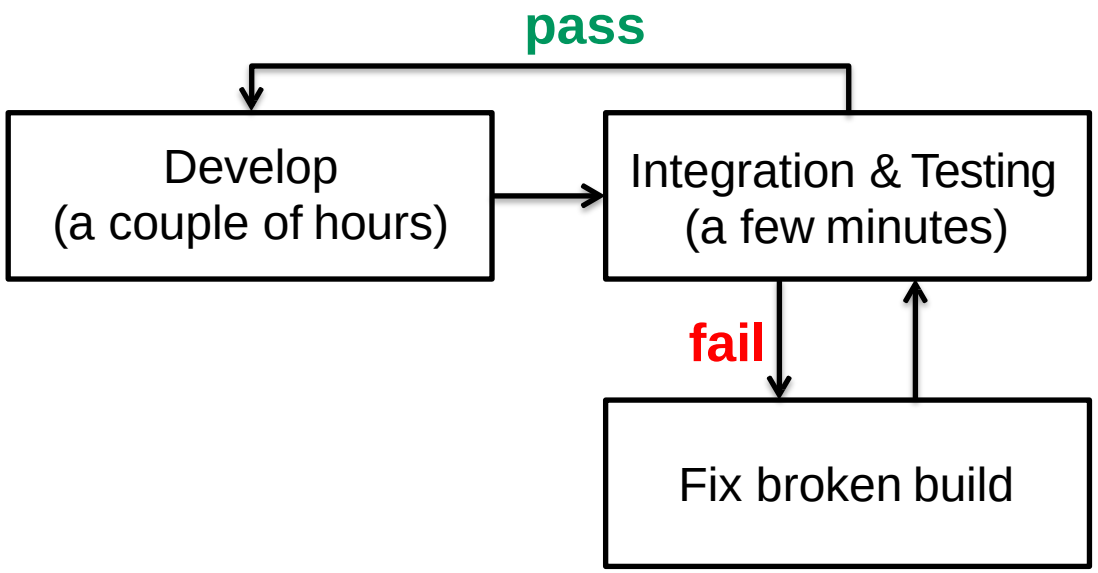
EXTREME PROGRAMMING AND CONTINUOUS INTEGRATION

"NO CODE SITS UNINTEGRATED FOR MORE THAN **A COUPLE OF HOURS**."

"At the end of every development episode, the code is integrated and **all the tests** must run at 100%."

"You need **a reasonably complete** test suite that runs in **a few minutes**."

CONTINUOUS INTEGRATION



2006: CONTINUOUS INTEGRATION EXTENDED

- Single source repository
- Integration machine
- Fix broken builds immediately
- Keep the build fast (10 minutes)
- Test in a clone of the production environment
- Automate deployment

http://www.martinfowler.com/articles/continuousIntegration.html

2009: CONTINUOUS DEPLOYMENT

“The high level of our process is dead simple: **Continuously integrate** (commit early and often). On commit automatically **run all tests**. If the tests pass **deploy to the cluster**. If the deploy succeeds, repeat.”

http://timothyfitz.com/2009/02/08/continuous-deployment/
http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

CONTINUOUS DEPLOYMENT

“So what magic happens in our test suite that **allows us to skip having a manual Quality Assurance** step in our deploy process? The magic is in the **scope, scale and thoroughness**. It’s a thousand test files and counting. **4.4 machine hours of automated tests** to be exact.”

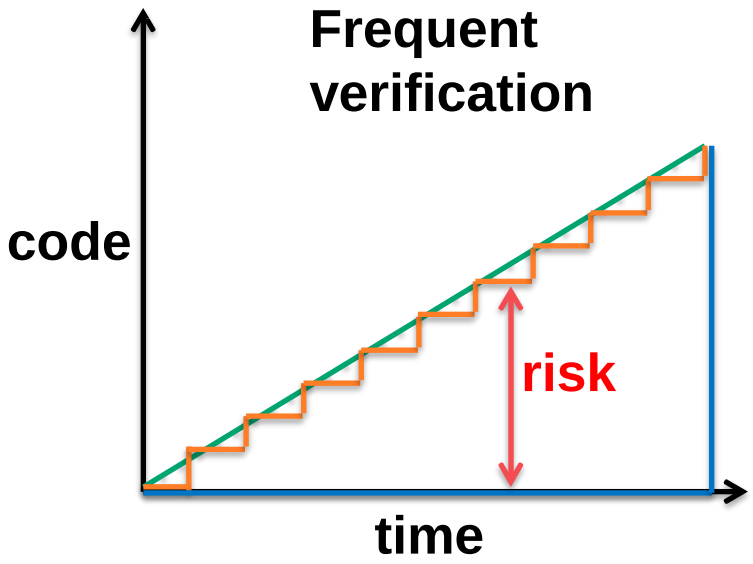
“Great test coverage is not enough. Continuous Deployment requires much more than that. Continuous Deployment means running all your tests, all the time. That means **tests must be reliable**.”

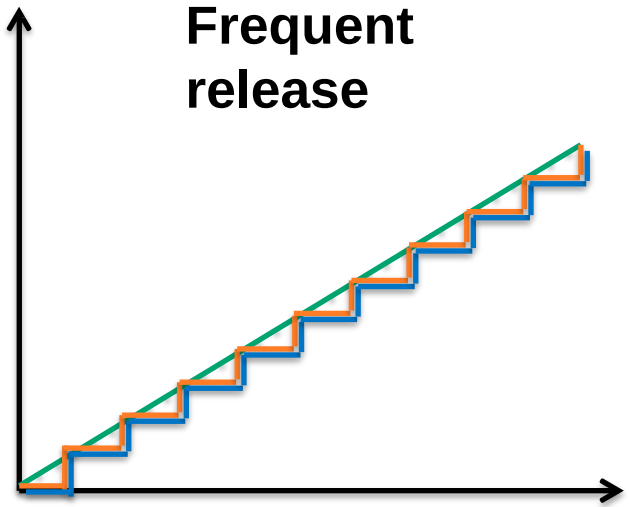
CONTINUOUS DEPLOYMENT

- **Context of cloud systems**
- **Problems in production will always happen**
- **How to mitigate the problems?**
  - Smaller releases have smaller scope and are easier to debug
  - Automated deployments allow fast fixing
  - Deploy to a subset of users first to mitigate problem scope
- **Production can be monitored and reverted automatically**

CONTINUOUS DEPLOYMENT

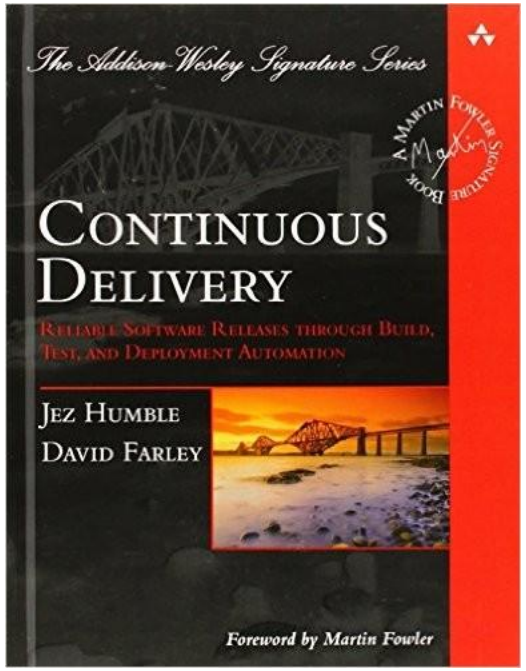
- Minimizes release risk
- Easier defect diagnosis
- Improved morale





2010: CONTINUOUS DELIVERY

- Every change should be releasable…
- ...but not necessarily released automatically
- Allows human verification



Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.

CONTINUOUS DELIVERY

You're doing continuous delivery when:

- Your software is **deployable** throughout its lifecycle
- Your team **prioritizes** keeping the software deployable over working on new features
- Anybody can get fast, **automated feedback** on the production readiness of their systems any time somebody makes a change to them
- You can perform **push-button deployments** of any version of the software to any environment on demand

http://martinfowler.com/bliki/ContinuousDelivery.html

SUMMARY

All models have similar ideas:

-Collect information before design

-Design before implementation

-Validate direction and quality during implementation

-Test before release

RE: FROM REQUIREMENTS TO IMPLEMENTATION