10th Advanced School on PARALLEL COMPUTING

Software engineering for HPC

Paolo Ciancarini - paolo.ciancarini@unibo.it
Department of Informatics - University of Bologna

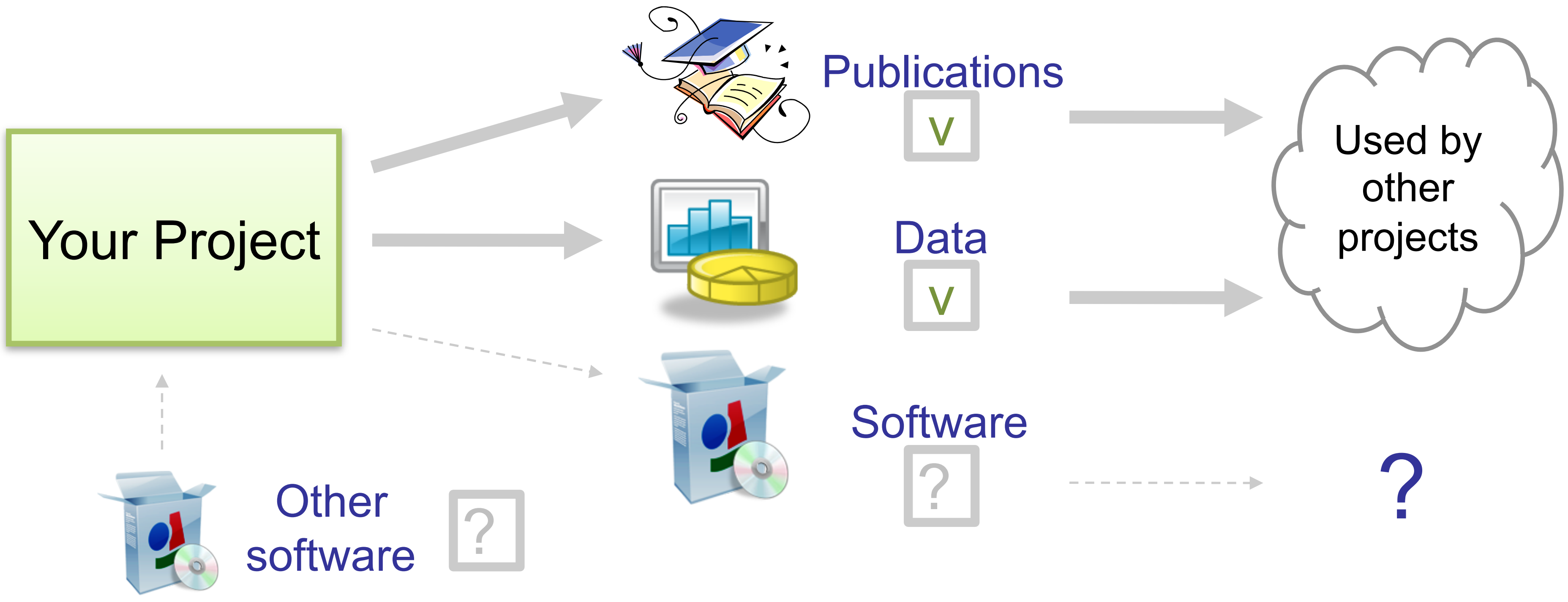
CINECA

cini consorzio interuniversitario nazionale per l'informatica

Agenda

- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
- Methods and tools
- Software processes

Using software in a research project



What is the future of your software?

A survey on scientists using HPC

- Scientists use more than 40% of their time using or building software
- 96% are self-taught
- No data on how well they do things (only 47% know the basics of testing)
- No data on how much software they reuse, or produce that is reused

[G. Wilson, Software Carpentry]

HPC

- Usage of supercomputers or advanced computing systems in solving complex scientific problems based on computationally intensive tasks
- It includes using
  - Infrastructure
    - HPC hardware architectures
    - Interconnection network
    - Operating systems and middleware
  - Parallel programming
    - Languages
    - Algorithms

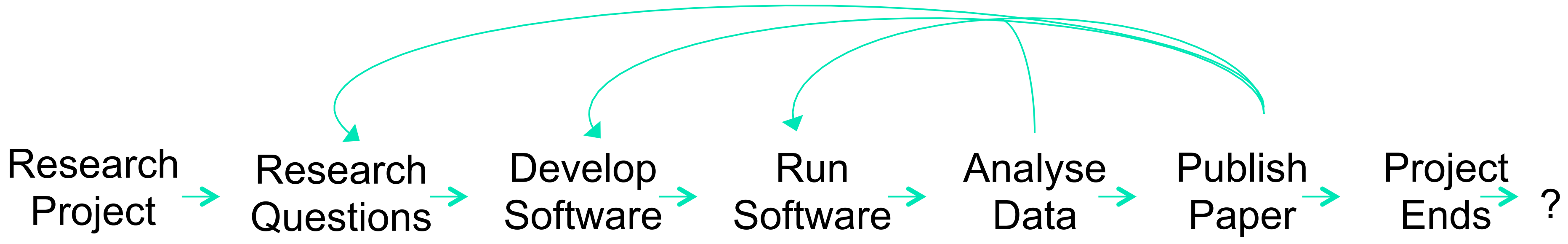
HPC developers

- The majority of the HPC developers are not sw engineers or computer scientists
- Primary goal of HPC developers: computational science
  - Achieving scientific results in some field of research using HP computers
- Hence, most HPC users build software, but almost all of them lack formal software engineering knowledge

HPC developers

- Most computational scientists learn software development from mentors or peers within their disciplines
  - Little or no formal training in computer science; informal “self-study” of programming and sw development is variable
  - Reward structure favors communicating *scientific* results, not the *software* that enabled the results
  - the software is often the basis of a competitive advantage
  - Horizontal knowledge transfer diffusion is slow

Typical software development in HPC



What happens to the software?

- Thrown away
- Kept on some systems, possibly in different versions
- Dumped on a code repository

What happens when…

- You have a follow-on project?
- Someone wants to (re)use the code?
- Someone wants to reproduce your results?
- Maintenance or future reuse should be considered?

Beware of software aging!

Software can *age*

- Ill-conceived modifications
- Functional operation degrades over time
- It becomes unsustainable, unusable
- Lack of proper maintenance
- Infrastructure (os, libraries, language platform) evolves
- Some software types more susceptible

Enters Software Engineering

“Software engineering is the discipline concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use”

[Sommerville 2007]

Software Engineering

- “The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.” [Naur & Randell, 1968]

What is Software Engineering?

A naive view:

Problem Specification $\xrightarrow{\textit{coding}}$ Final Program

But ...

- Where did the *problem specification* come from?
- How do you know the problem specification corresponds to and satisfies the *user's needs*?
- How did you decide how to *structure* your program?
- How do you know the program actually *meets the specification*?
- How do you know your program will always *work correctly*?
- What do you do if the users' *needs change*?
- How do you *divide tasks up* if you have more than a one person in the developing team?
- How do you reuse exisiting software for solving similar problems?

Software Engineering

- Some definitions and issues
  - “state of the art of developing quality software on time and within budget”
- Trade-off between a system perfectly engineered and the available resources
  - SE has to deal with real-world issues
- State of the art
  - Community decides on “best practice” + life-long education

What is Software Engineering?

"multi-person construction of multi-version software"

— Parnas

- Team-work
  - Scale issue ("program well" is not enough) + Communication Issue
- Successful software systems must evolve or perish
  - Change is the norm, not the exception

What is Software Engineering?

“software engineering is <u>different</u> from other engineering disciplines”

— Sommerville

- Not constrained by physical laws
  - limit = human knowledge
- It is constrained by social forces
  - Balancing stake-holders needs
  - Consensus on functional and especially non-functional requirements

Software Engineering

- Software engineering is … dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build
- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software [Guide to the Software Engineering Body of Knowledge]

Roadmap

- What is Software Engineering?
- **The Iterative Development Lifecycle**
- Software Development Activities
- Methods and tools
- Software processes

Software Development Activities

Requirements Collection	Establish customer's needs
Analysis	Model and specify the requirements ("what")
Design	Model and specify a solution ("how")
Implementation	Construct a solution in software
Testing	Validate the software against its requirements
Deployment	Making a software available for use
Maintenance	Repair defects and adapt the sw to new requirements

NB: these are ongoing activities, not sequential phases!

First development step: requirements

- The first step in any development process consists in understanding the needs of someone asking for a software
- The needs should be stated explicitly in “requirements”, which are statements requiring some function or property to the final software system

A Software Requirement is … (IEEE 610)

1) A condition or capability needed by a user to solve a problem or achieve an objective
2) A condition or capability that must be met or possessed by a system or component to satisfy a contract, standard, specification, or other formally imposed documents
3) A documented representation of a condition or capability as in (1) or (2)

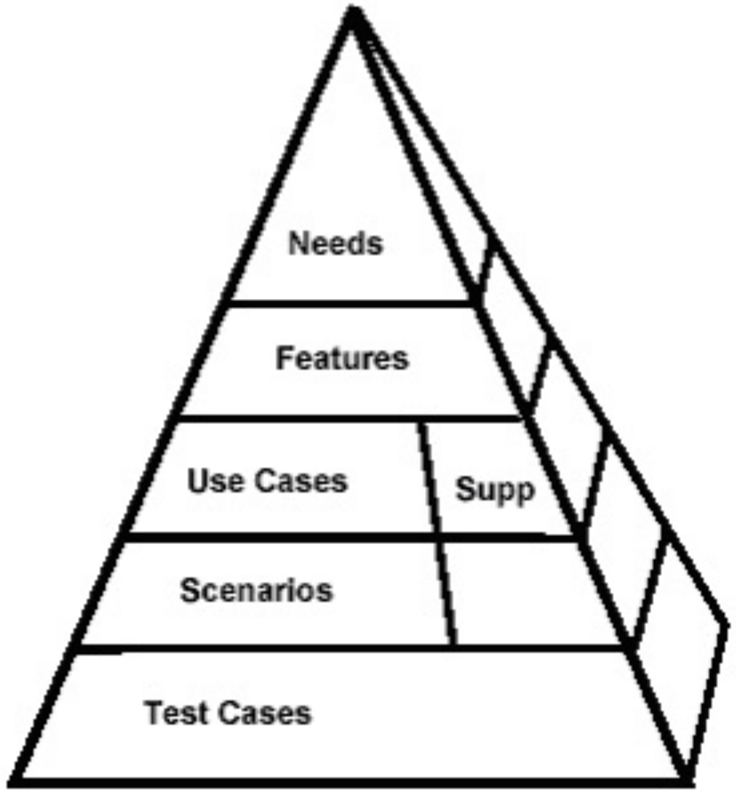
The requirements pyramid

Some user has some need

Needs are answered by “features” that some system must have

Each feature corresponds to a need and is a collection of requirements

Features and requirements can be aggregated in “scenarios” where testing can prove that the features will satisfy the needs

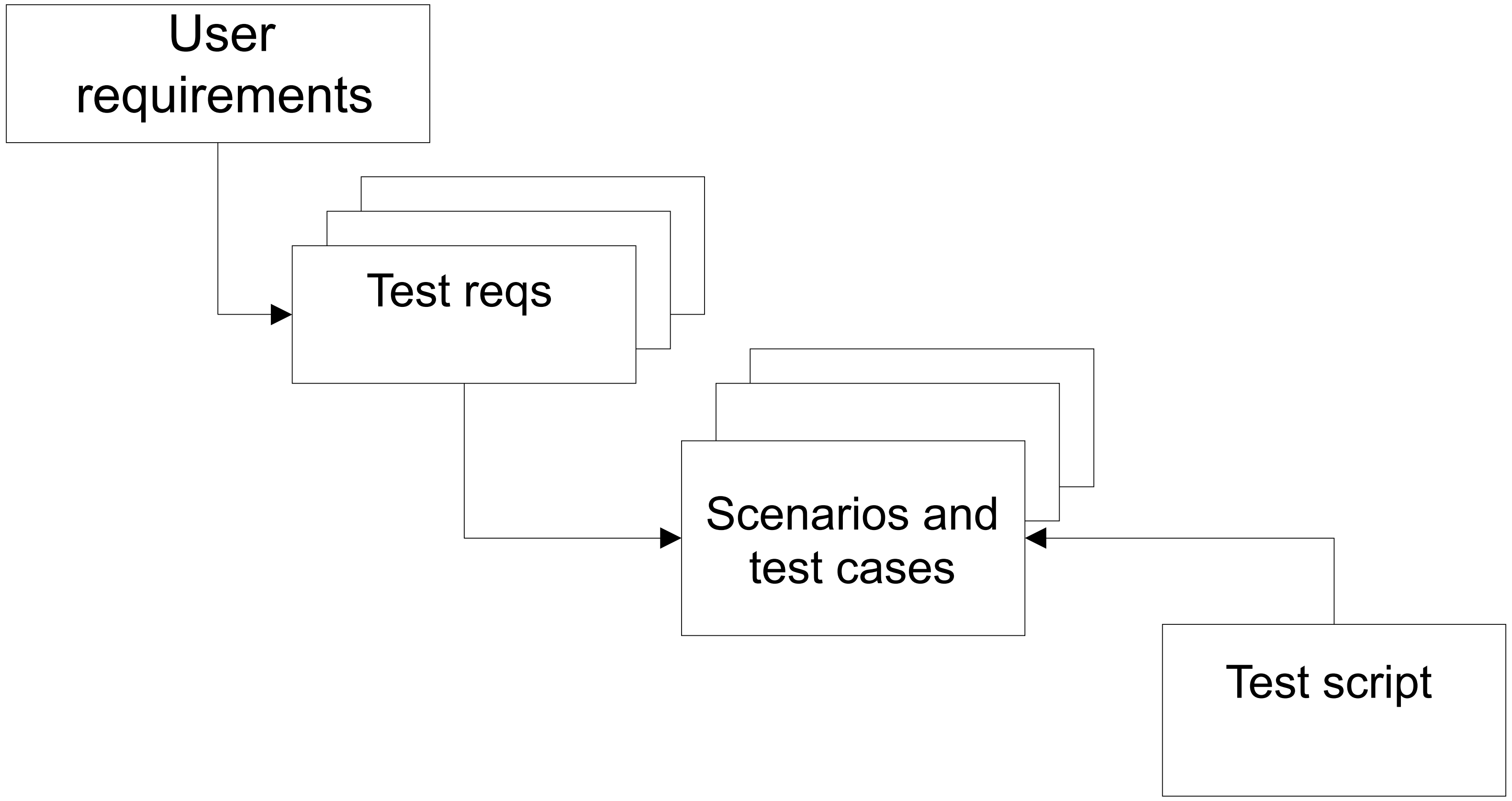


www.ibm.com/developerworks/rational/library/04/r-3217

Sw requirement specification

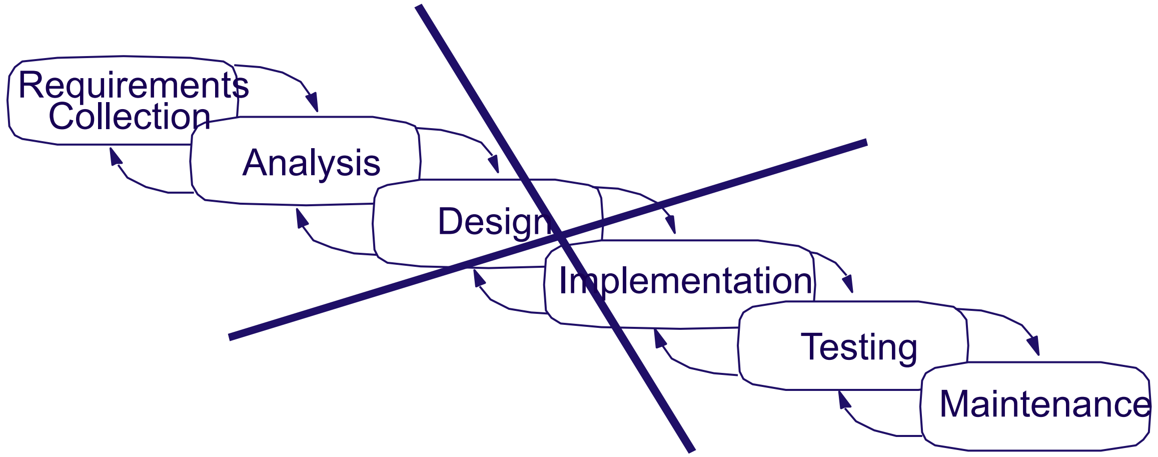
- An SRS document is an organization’s understanding of the requirements and dependencies of some system
- It is written (usually) prior to any actual design or development work
- It is a two-way insurance policy that assures that both the client and the organization understand the other’s requirements from that perspective at a given point in time
- The SRS document itself states those functions and capabilities a software system (e.g., an app, an eCommerce Web site, etc) must provide, as well as it states any required constraints by which the system must abide

Requirements and tests



The Classical Software Lifecycle

The classical software lifecycle models the software development as a step-by-step "waterfall" between the various development phases



The waterfall model is unrealistic for many reasons:

- **Requirements must be *frozen too early* in the life-cycle**
- **Requirements are *validated too late***
- **Risks in costructing wrongly the software are high**

Problems with the waterfall lifecycle

1. “Real projects rarely follow the sequential flow that the waterfall model proposes. *Iteration* always occurs and creates problems in the application of the paradigm”

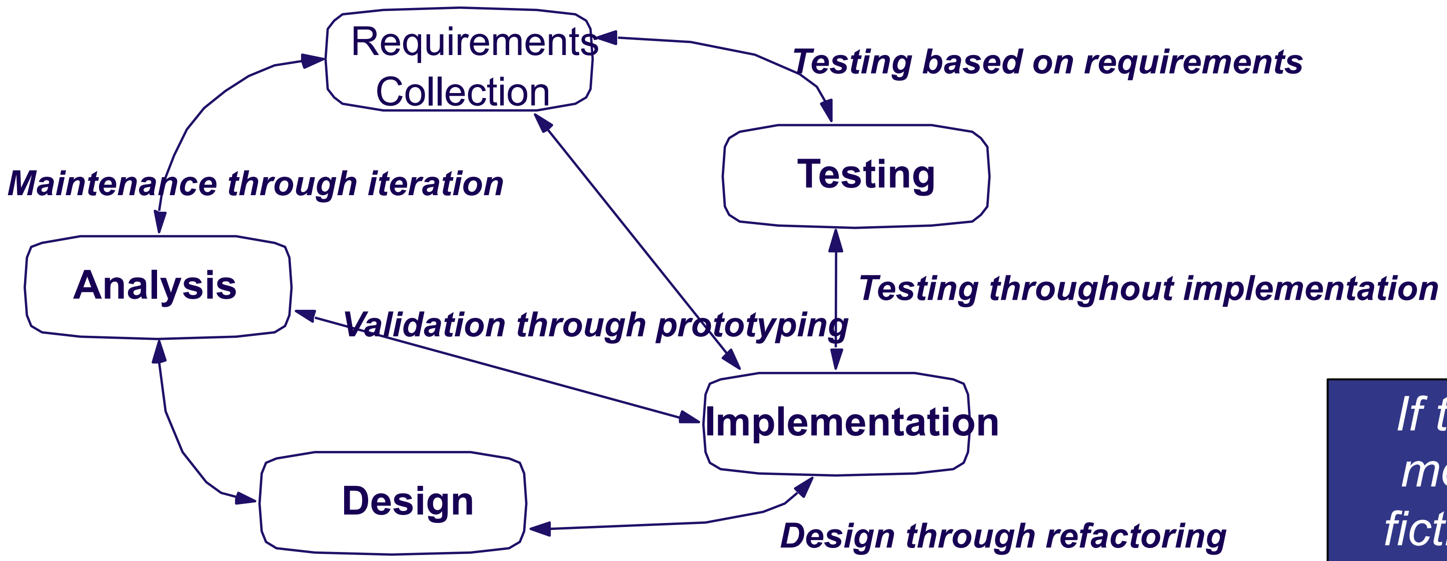
2. “It is often *difficult* for the customer *to state all requirements* explicitly. The classic life cycle requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.”

3. “The customer must have patience. A *working version* of the program(s) will not be available until *late in the project* timespan. A major blunder, if undetected until the working program is reviewed, can be disastrous.”

— *Pressman, SE, p. 26*

Iterative Development

In practice, development is always iterative,
and *most* activities can progress in parallel


Requirements Collection
Testing based on requirements
Testing
Maintenance through iteration
Analysis
Testing throughout implementation
Validation through prototyping
Implementation
Design
Design through refactoring

If the waterfall model is pure fiction, why is it still the dominant software process?

Iterative Development

- Plan to iterate your analysis, design and implementation
  - You will not get it right the first time, so integrate, validate and test as frequently as possible
  - During software development, more than one iteration of the software development cycle may be in progress at the same time
  - This process may be described as an 'evolutionary acquisition' or 'incremental build' approach

Incremental Development

Plan to *incrementally* develop (i.e., prototype) the system

- If possible, *always have a running version* of the system, even if most functionality is yet to be implemented
- *Integrate* new functionality as soon as possible
- *Validate* incremental versions against user requirements.

Roadmap

- What is Software Engineering?
- The Iterative Development Lifecycle
- **Software Development Activities**
- Methods and tools
- Software processes

Requirements Collection

User requirements are often expressed *informally*:

- features
- usage scenarios

Although requirements may be documented in written form, they may be *incomplete*, *ambiguous*, or *incorrect*

Changing requirements

Requirements *will* change!

- *inadequately captured* or expressed in the first place
- user and business *needs may change* during the project

Validation is needed *throughout* the software lifecycle, not only when the “final system” is delivered!

- build constant *feedback* into your project plan
- plan for *change*
- early *prototyping* [e.g., UI] can help clarify requirements

Requirements Analysis

Analysis is the process of specifying *what* a system will do

- The goal is to provide an understanding of what the system is about and what its underlying concepts are

The result of analysis is a *specification document*

Does the requirements specification correspond to the users' actual needs?

Object-Oriented Analysis

An *object-oriented analysis* results in models of the system which describe:

- *classes* of objects that exist in the system
  - *responsibilities* of those classes
- *relationships* between those classes
- *use cases* and *scenarios* describing
  - *operations* that can be performed on the system
  - allowable *sequences* of those operations

Prototyping

A *prototype* is a software program developed to test, explore or validate a hypothesis, i.e. *to reduce risks*

An *exploratory prototype*, also known as a *throwaway prototype*, is intended to *validate requirements* or *explore design choices*

- UI prototype — validate user requirements
- rapid prototype — validate functional requirements
- experimental prototype — validate technical feasibility

Prototyping

An *evolutionary prototype* is intended to evolve in steps into a finished product.

- iteratively “grow” the application, *redesigning* and *refactoring* along the way

First do it,
then do it right,
then do it fast.

Design

Design is the process of specifying *how* the specified system behaviour will be realized from software components. The results are *architecture* and *detailed design documents*.

Object-oriented design delivers models that describe:

- how system operations are implemented by *interacting objects*
- how classes refer to one another and how they are related by *inheritance*
- *attributes* and *operations* associated to classes

Design is an iterative process, proceeding in parallel with implementation!

Conway’s Law

- The law: *Organizations that design systems are constrained to produce designs that are copies of the communication structures of these organizations*
- Example: "If you have four groups working on a compiler, you'll get a 4-pass compiler”
- Several studies found significant differences in modularity when software is outsourced, consistent with a view that distributed teams tend to develop more modular products

Implementation and Testing

Implementation is the activity of *constructing* a software solution to the customer's requirements.

Testing is the process of *validating* that the solution meets the requirements.

- The result of implementation and testing is a *fully documented* and *validated* solution.

Testing, Testing!

1

- Provide automated build process
  - Far easier & quicker to validate changes
  - e.g. Make, Ant, Maven

2

- Provide automated regression test suite - TDD
  - Do changes break anything?
  - JUnit, CPPUnit, xUnit, fUnit, …

3

- Join together: automated build & test
  - A ‘fail-fast’ environment

4

- Infrastructure support
  - Nightly builds – run build & test overnight, send reports
  - Continuous integration - run build & test when codebase changes

Towards *anytime releasable* code!

Design, Implementation and Testing

Design, implementation and testing are iterative activities

- The implementation does not “implement the design”, but rather the design document *documents the implementation*!

- System tests reflect the requirements specification
- Testing and implementation go hand-in-hand
  - Ideally, test case specification *precedes* design and implementation

Maintenance

Maintenance is the process of changing a system after it has been deployed.

- *Corrective maintenance*: identifying and repairing *defects*
- *Adaptive maintenance*: *adapting* the existing solution to new platforms
- *Perfective maintenance*: implementing *new requirements*
- *Preventive maintenance*: repairing a software product before it breaks

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered "maintenance"!

Maintenance activities

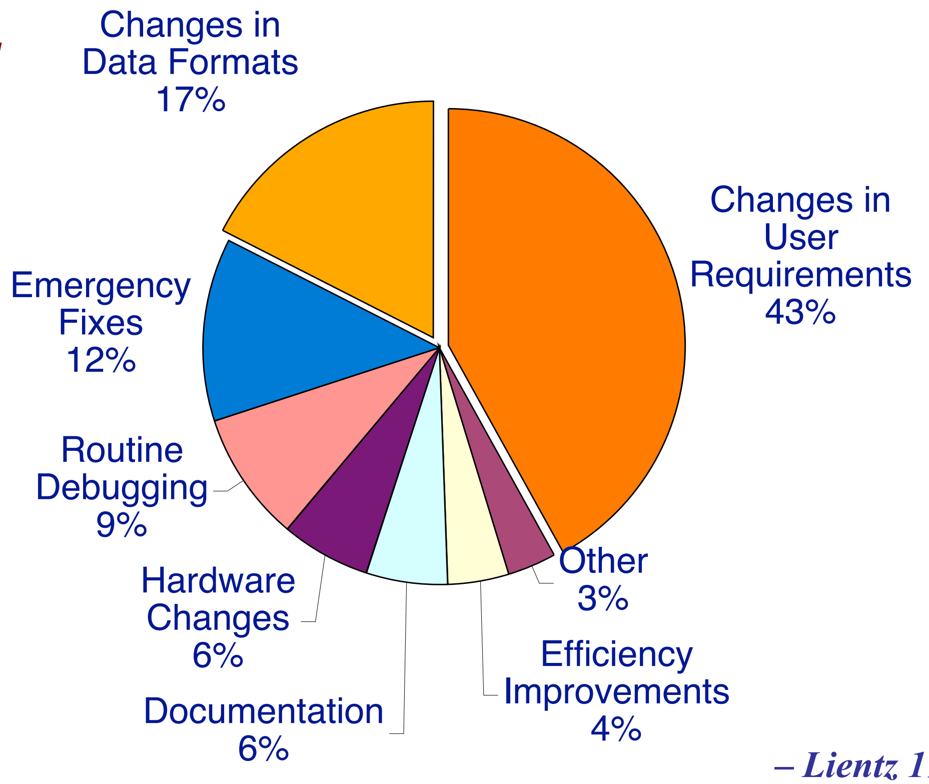
“Maintenance” entails:

- configuration and version management
- reengineering (redesigning and refactoring)
- updating all analysis, design and user documentation

Repeatable, automated tests enable evolution and refactoring

Maintenance costs

"Maintenance"
typically accounts for
70% of software costs!



***Means:* most project costs concern continued development *after* deployment**

– Lientz 1979

Configuration management

- Run your own CM system, if you have the resources
  - Generally easy to set up
  - Full control, but be sure to back it up!
- Some public solutions can offer most of these for free
  - SourceForge, GoogleCode, GitHub, Codeplex, Launchpad, Assembla, Savannah, …
  - BitBucket for private code base (under 5 users)
  - See (for hosting code and related tools) http://software.ac.uk/resources/guides/choosing-repository-your-software-project
  - See (for hosted continuous integration) http://www.software.ac.uk/blog/2012-08-09-hosted-continuous-integration-delivering-infrastructure

"If you're not using version control, whatever else you might be doing with a computer, it's not science" – Greg Wilson, Software Carpentry

Deployment

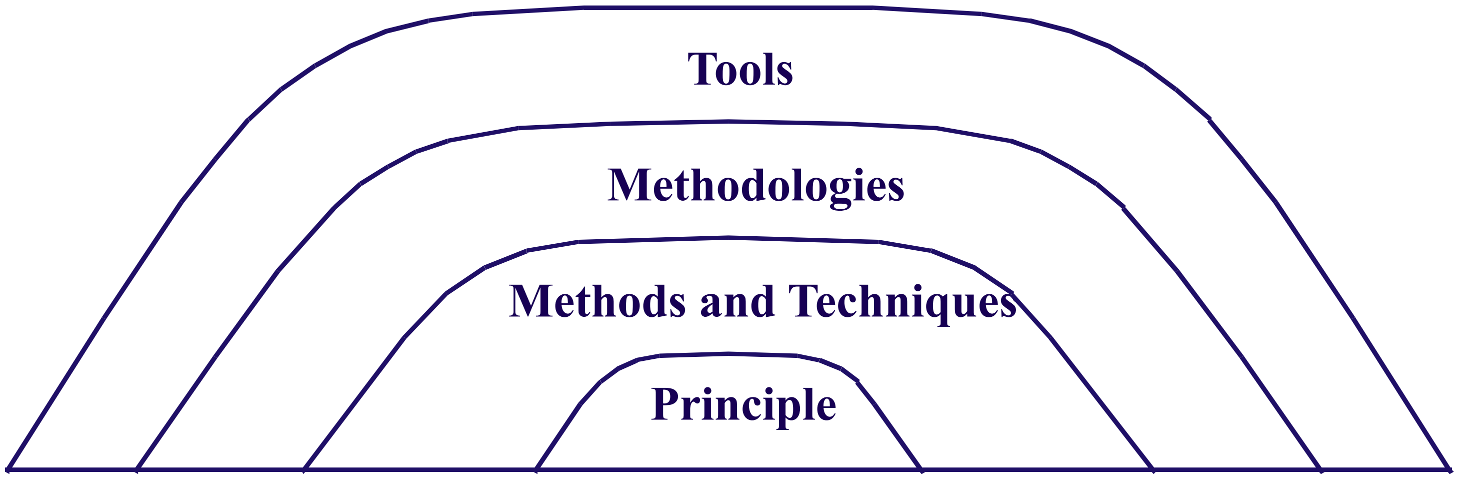
- Virtual Machines
  - Software pre-installed, ready to run
  - Often easiest
  - Not enough in itself – documentation!
- Release software
  - Prioritise & select requirements -> Develop -> Test -> Commit changes to repository -> Test -> Release
  - Documentation (minimum: quick start guide)
- Licencing
  - Specify rights for using, modifying and redistributing

Roadmap

- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
- **Methods and tools**
- Software processes

Principles, methods and tools

Principle **= general statement describing desirable properties**
Method **= general guidelines governing some activity**
Technique **= more technical and mechanical than method**
Methodology **= package of methods and techniques packaged**



— Ghezzi et al. 1991

Object-Oriented Methods: a short history

First generation:

- Adaptation of existing notations (ER diagrams, state diagrams ...): Booch, OMT, Shlaer and Mellor, ...
- Specialized design techniques:
  - CRC cards; responsibility-driven design; design by contract

Second generation:

- Fusion: Booch + OMT + CRC + formal methods

Third generation:

- Unified Modeling Language:
  - uniform notation: Booch + OMT + Use Cases + ...
  - various UML-based methods (e.g. Catalysis)

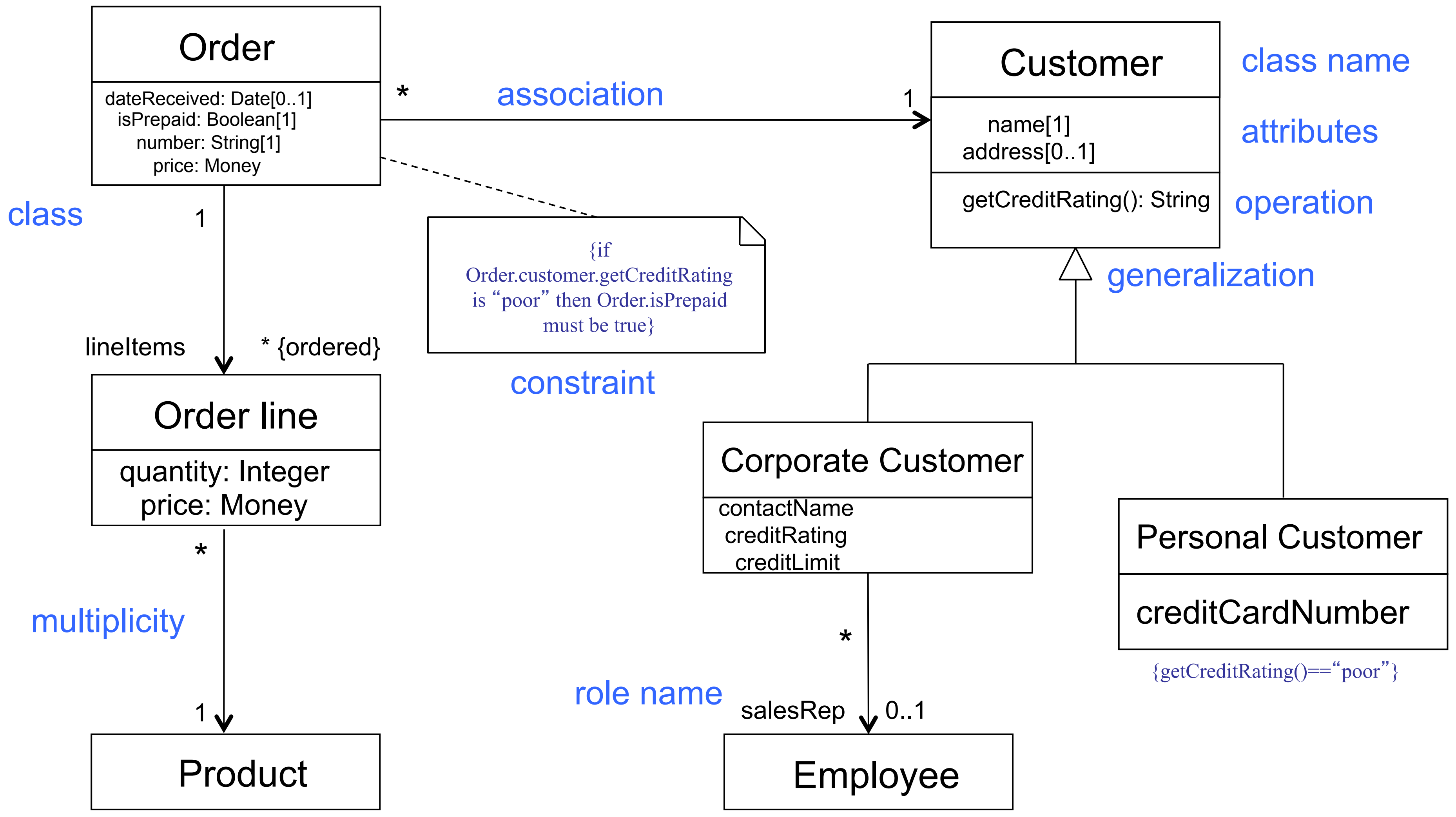
UML is a modeling language

- A modeling language allows the specification, the visualization, and the documentation of the development of a software system
- The models are *artifacts* which clients and developers use to communicate
- UML 1.* is a modeling language
- UML 2.* is also a programming language

Meaning of models for software

- A model is a *description* of the structure and meaning of a system
- A model is always an *abstraction* at some level: it captures the essential aspects of a system and ignores some details
- NB: a model can be also *generator* of potential *configurations* of systems

Class diagram: example



Roadmap

- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
- Methods and tools
- **Software processes**

Software: the product of a process

- Many kinds of software products → many kinds of development processes
- “Study the process to improve the product”

- A software development process can be described according to some specific “model”
- Examples of process models: waterfall, iterative, agile, extreme,…
- The models differ mainly in the roles and activities that the stakeholders cover

The production of software in the contemporary world

- Centralized vs distributed (offshore)
- Code centric vs data intensive
- In lab vs in the large
- Long planned vs continuous release
- Single system vs product lines
- Stakeholders: community vs market

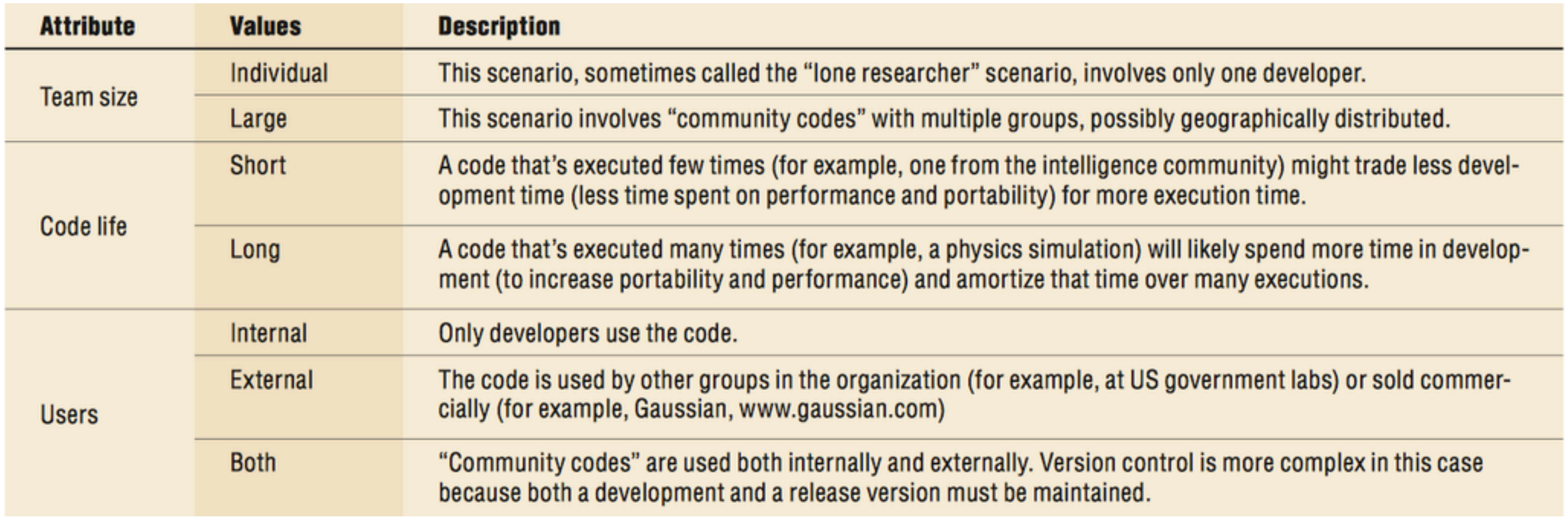
Stakeholders

Typical stakeholders in a sw process

- Designers
- Management
- Technicians
- Decisors
- Users
- Funding people
- …

Each stakeholder has a specific viewpoint on the product and its development process

HPC stakeholders attributes



Attribute	Values	Description
Team size	Individual	This scenario, sometimes called the "lone researcher" scenario, involves only one developer.
	Large	This scenario involves "community codes" with multiple groups, possibly geographically distributed.
Code life	Short	A code that's executed few times (for example, one from the intelligence community) might trade less development time (less time spent on performance and portability) for more execution time.
	Long	A code that's executed many times (for example, a physics simulation) will likely spend more time in development (to increase portability and performance) and amortize that time over many executions.
Users	Internal	Only developers use the code.
	External	The code is used by other groups in the organization (for example, at US government labs) or sold commercially (for example, Gaussian, www.gaussian.com)
	Both	"Community codes" are used both internally and externally. Version control is more complex in this case because both a development and a release version must be maintained.

Lifecycles

- Lifecycles are created and adapted to different products
- Both products and lifecycles can be evaluated for their quality
- Software needs several different lifecycle layers:
  - Industrial lifecycle
  - Development lifecycle (reqs, design, build, test)
  - Operational lifecycle (eg.: portal)

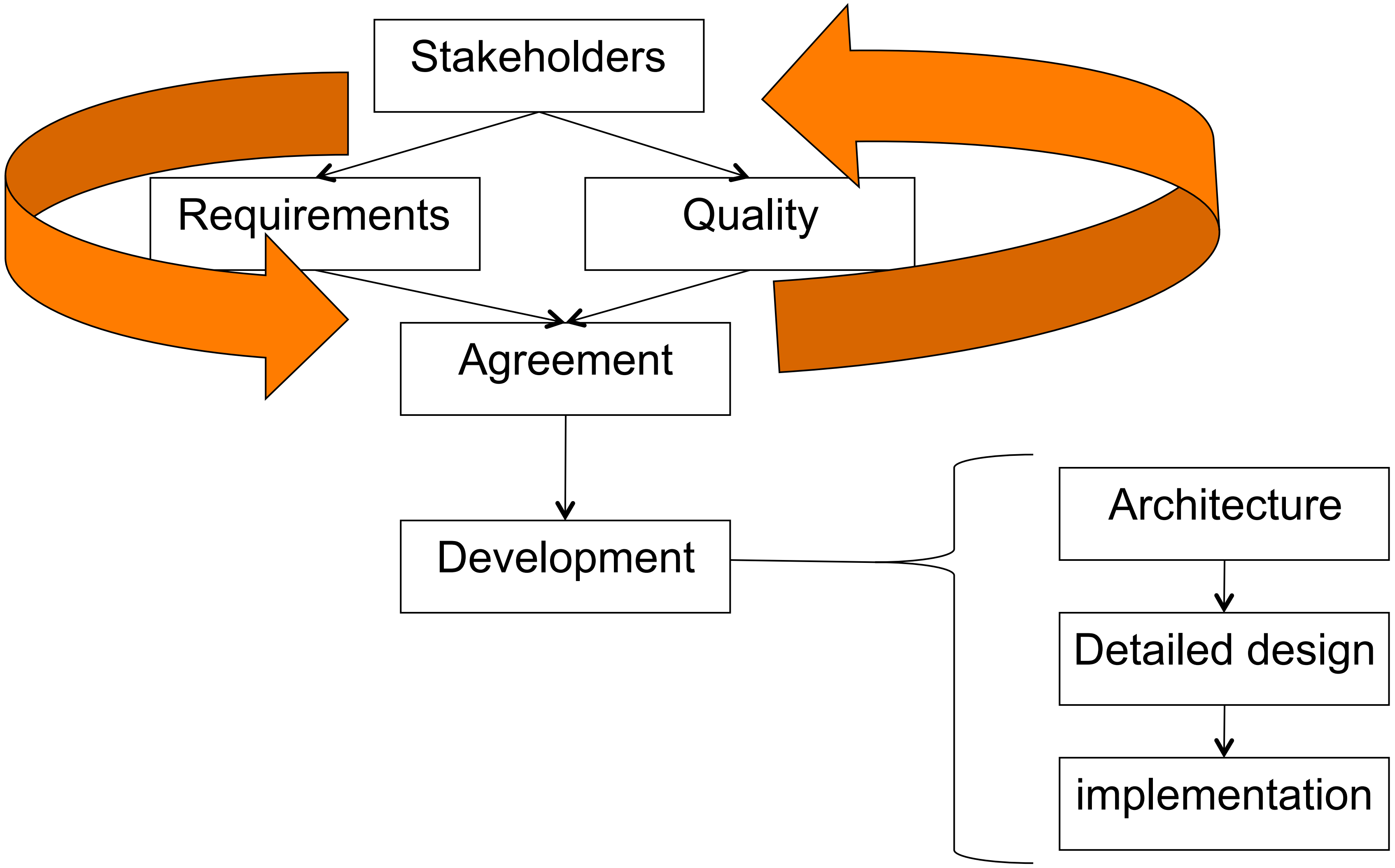
The software development process

- **Software process**: set of roles, activities, and artifacts necessary to create a software product
- Possible roles: designer, developer, tester, maintenance, ecc.
- Possible artifacts: source code, executables, specifications, comments, test suite, etc.

Activities

- Each organization differs in products it builds and the way it develops them; however, most development processes include:
  - Specification
  - Design
  - Verification and validation
  - Evolution
- The development activities must be modeled to be managed and supported by automatic tools

The lifecycle, traditional



Control variables of sw processes

- Time – duration of the project
- Quality – satisfying the stakeholders
- Resources – personnel, equipment, etc.
- Scope – what is to be done; the features to be implemented

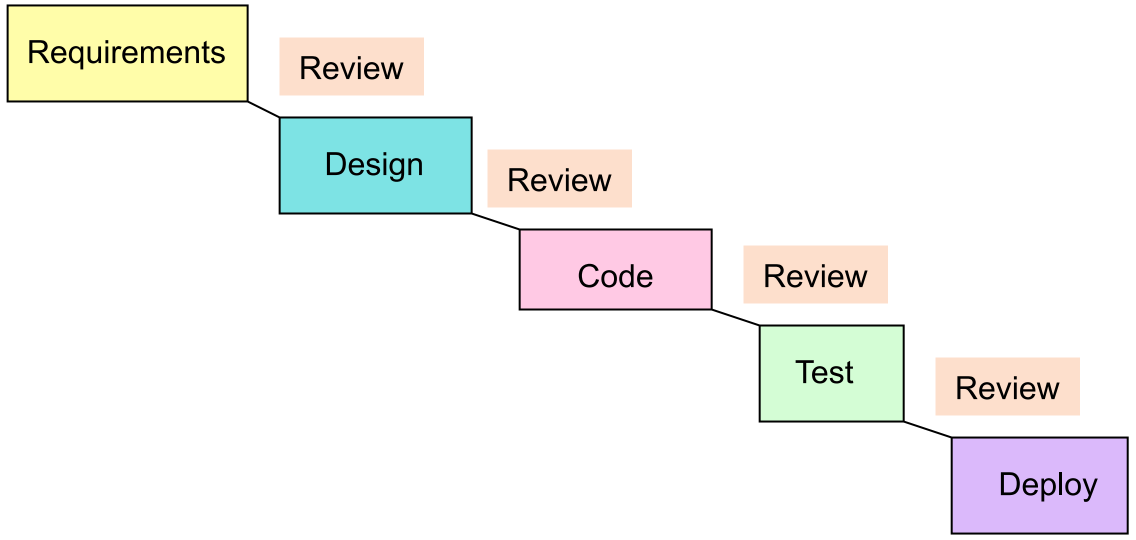
- These control variables are very difficult to control all; the simplest and most effective to control is scope
- The theory of software process models has the goal of controlling the other variables as well

Models for the software process

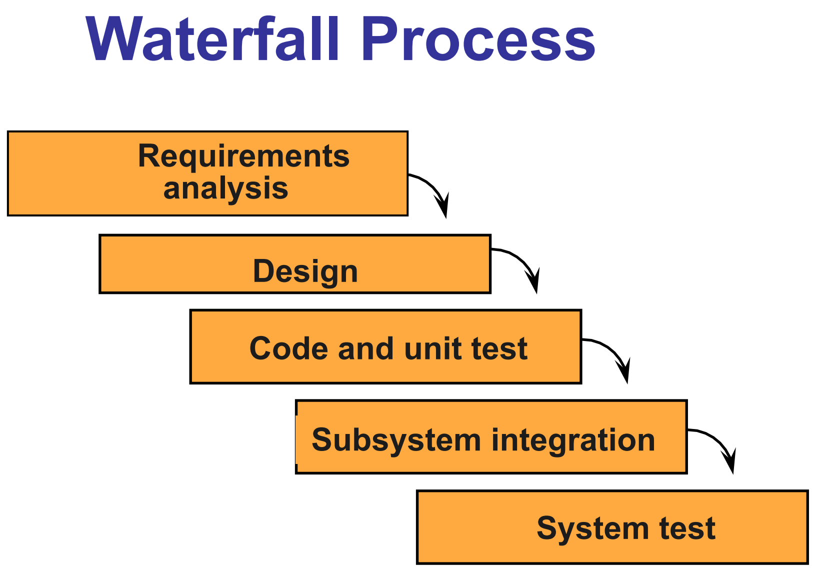
- Waterfall (planned, linear)
- Spiral (planned, iterative)
- Agile (unplanned, test driven)

Traditional process model

- Waterfall



Waterfall characteristics



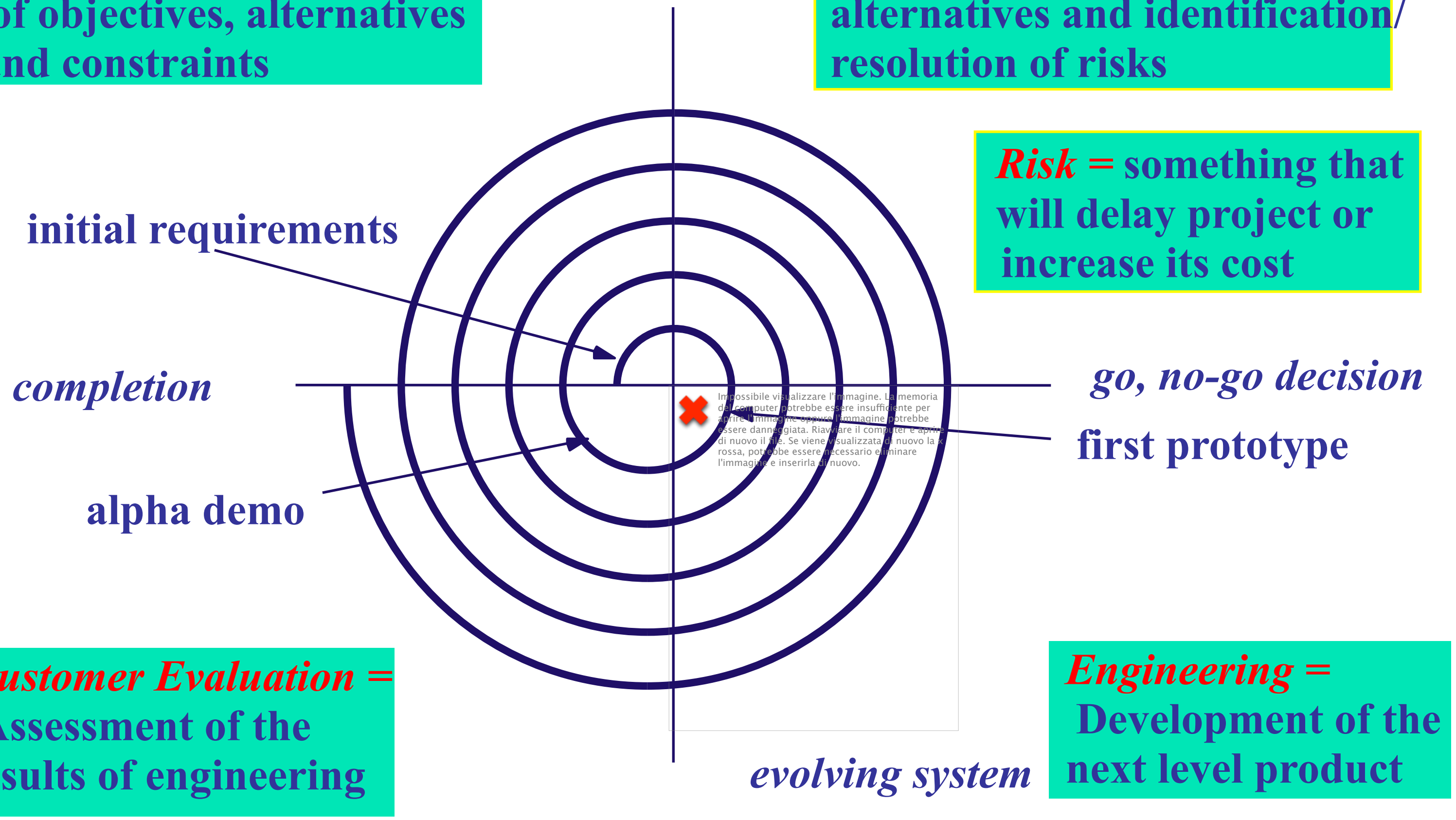
- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations

The Spiral Lifecycle (B.Bohem)

Planning = determination of objectives, alternatives and constraints

Risk Analysis = Analysis of alternatives and identification/ resolution of risks

Risk = something that will delay project or increase its cost

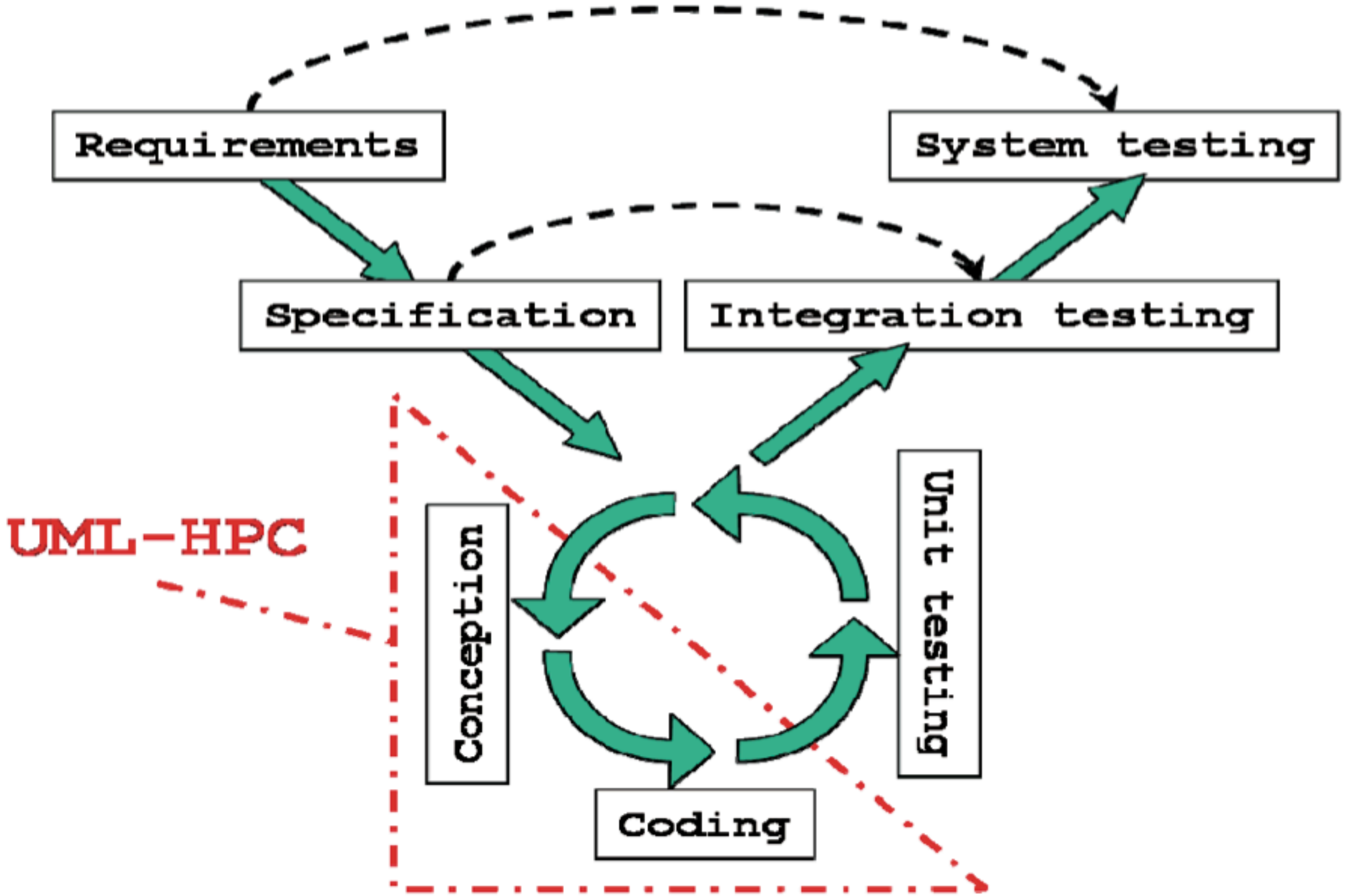


Impossibile visualizzare l'immagine. La memoria del computer potrebbe essere insufficiente per aprire l'immagine oppure l'immagine potrebbe essere danneggiata. Riavviare il computer e aprire di nuovo il file. Se viene visualizzata di nuovo la x rossa, potrebbe essere necessario eliminare l'immagine e inserirla di nuovo.

Customer Evaluation = Assessment of the results of engineering

Engineering = Development of the next level product

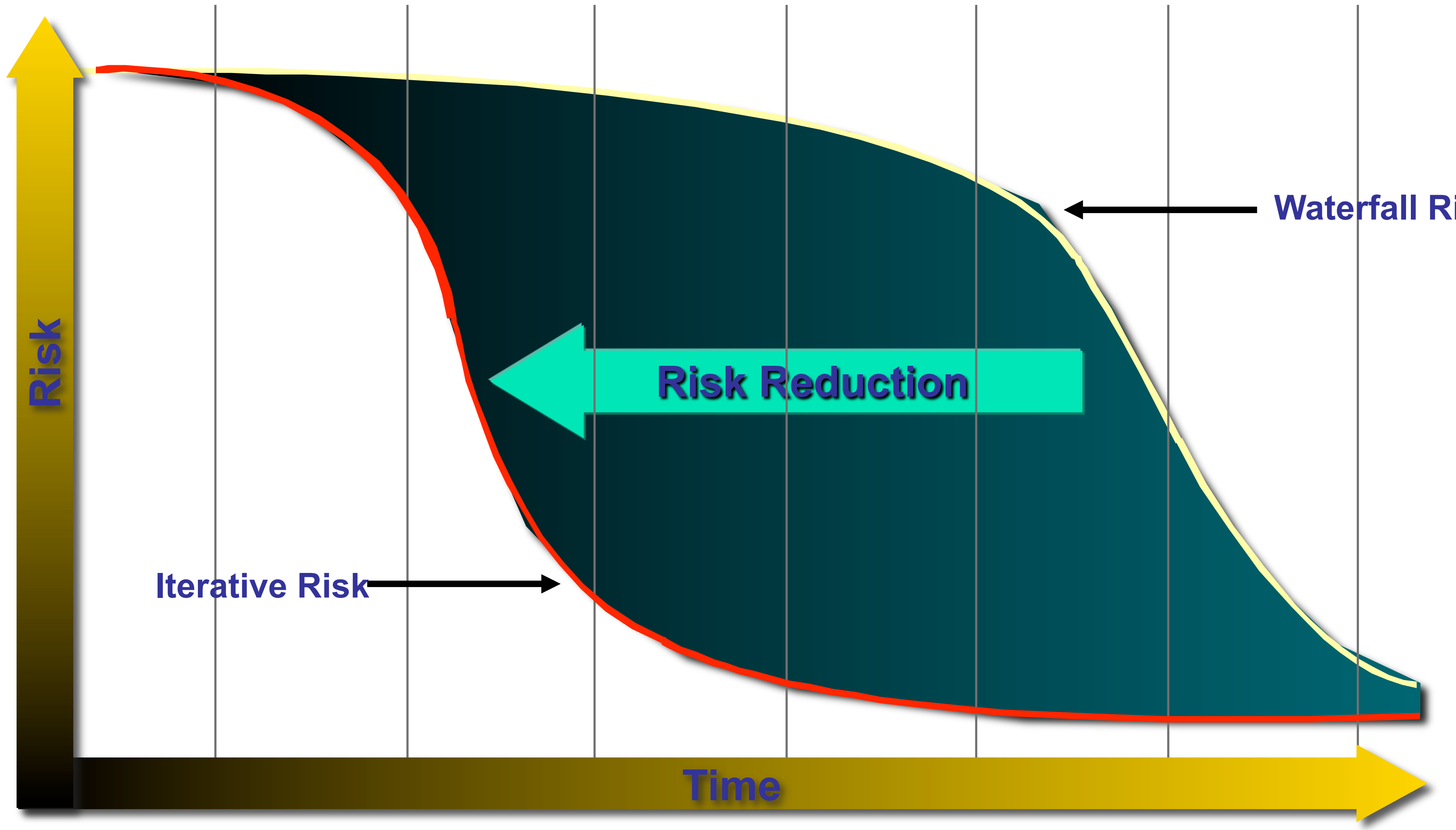
A process for HPC [Lugato 2010]



Iterative Development

- **Phase** - the time between two major project milestones, during which a well-defined set of objectives is met, artifacts are completed, and decisions are made to move or not into the next phase. A phase includes one or more iterations.
- **Iteration** – a time period in which a number of predefined tasks are performed and results are evaluated to feedback to the next iteration. An iteration results in a release.
- **Release (external)** – a coherent set of completed functionalities (code and other artifacts) that is useful to the stakeholders of the system
- **Release (internal)** - a coherent set of completed functionalities that is used only by the development organization, as part of a milestone, or for a demonstration to users

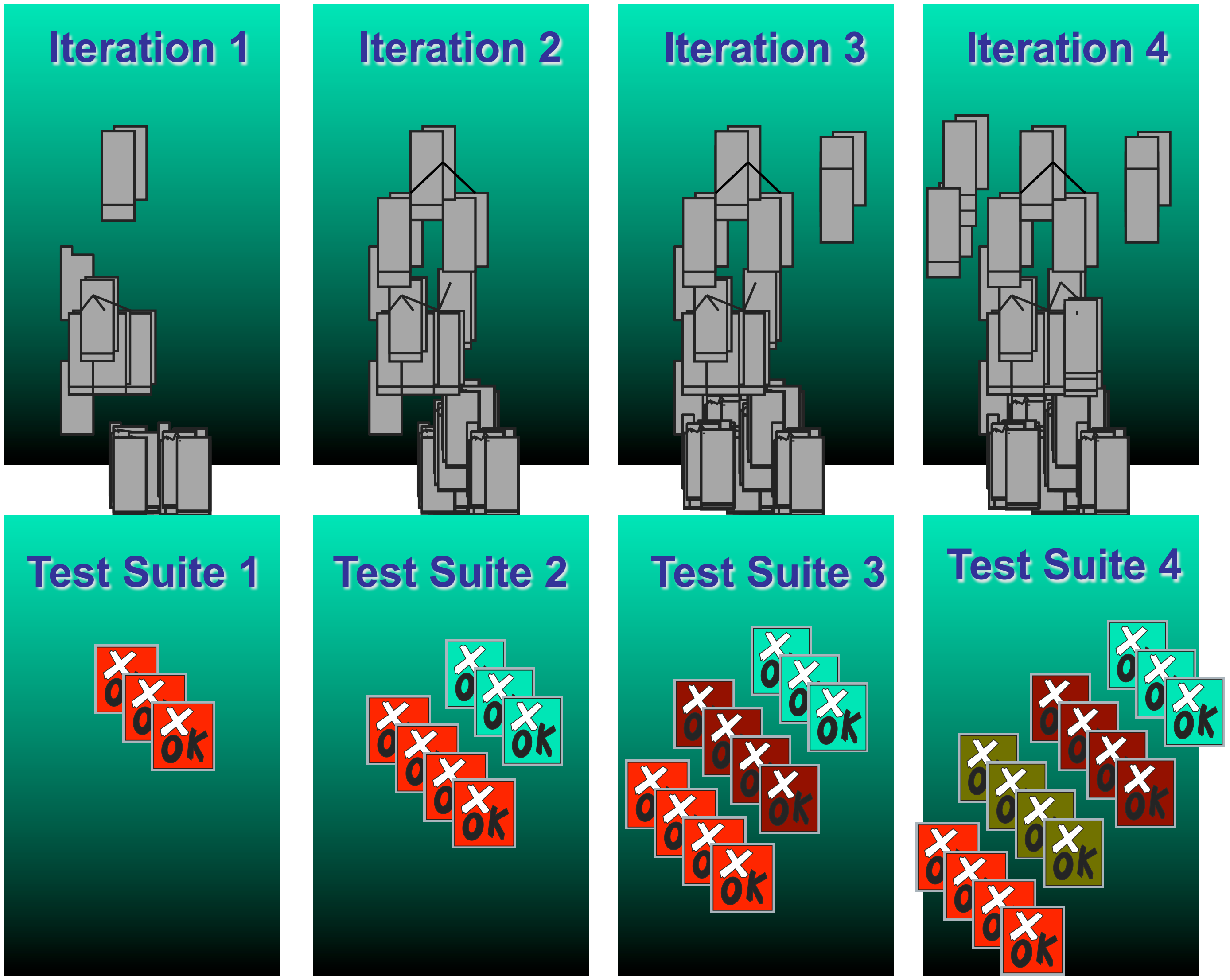
Risk: waterfall vs iterative



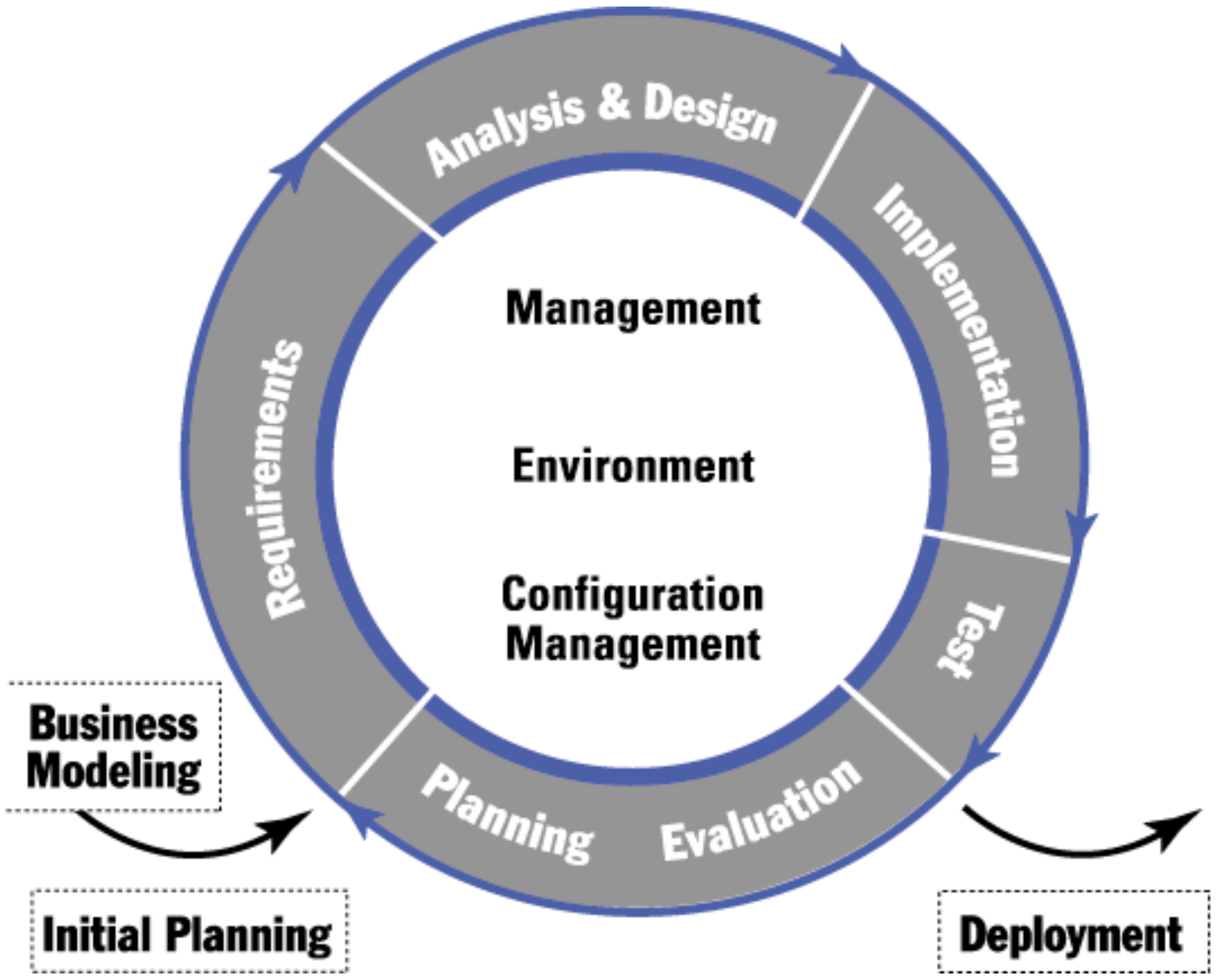
Test each iteration

UML Model and Implementation

Tests

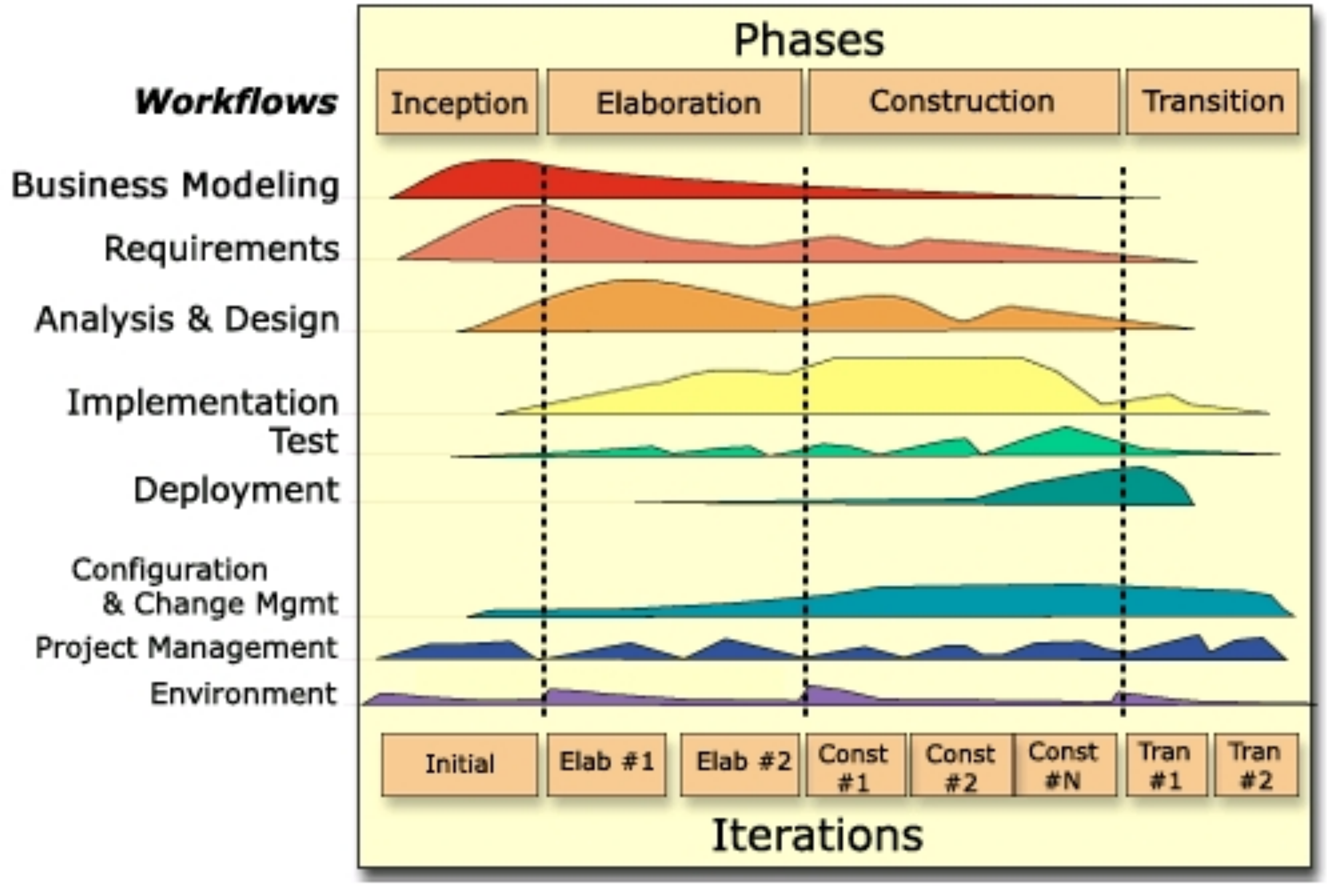


RUP is iterative



http://www-306.ibm.com/software/rational/sw-library/

Original RUP



Benefits of Iterative Development

A software project evolves in iterations to:

- Mitigate risk
- Accommodate change
- Learn along the way
- Improve the quality of the artifacts
- Exploit reuse thus increasing productivity

Committed vs involved

- The core roles in development teams are those **committed** to the project in the process - they are the ones producing the product: product owner, team, project manager
- The other roles in teams are those with no formal role and **infrequent involvement** in the process - and must nonetheless be taken into account

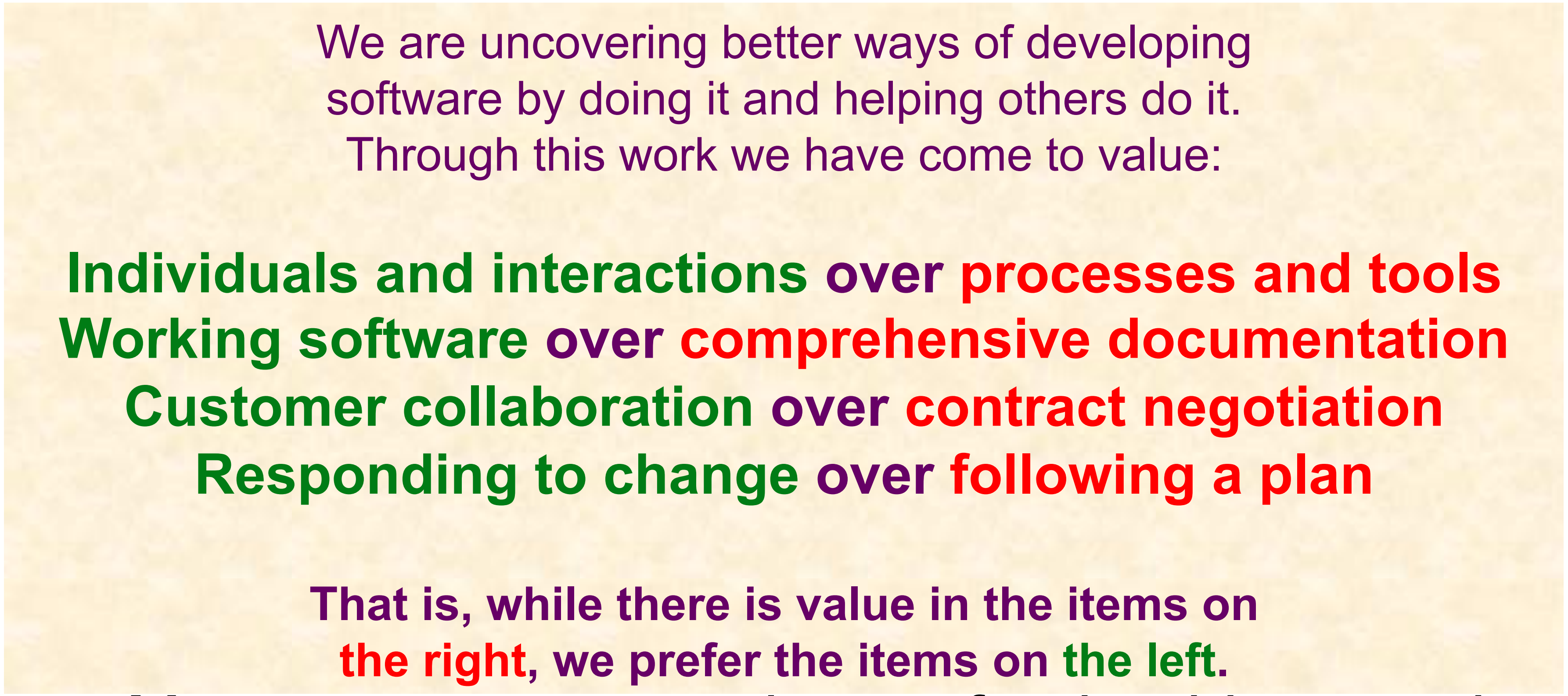
A story

A pig and a chicken are walking down a road.
The chicken looks at the pig and says, "Hey, why don't we open a restaurant?" The pig looks back at the chicken and says, "Good idea, what do you want to call it?" The chicken thinks about it and says, "Why don't we call it 'Ham and Eggs'?" "I don't think so," says the pig, "I'd be committed but you'd only be involved."

HEY PIG, I WAS THINKIN' WE SHOULD OPEN A RESTAURANT.
I DON'T KNOW. WHAT WOULD WE CALL IT?
Ham -n- Eggs
HOW ABOUT "HAM -N- EGGS."
NO THANKS, I'D BE COMMITTED, BUT YOU'D ONLY BE INVOLVED!
By Clark & Vizdos
© 2006 implementingscrum.com

Agile ethics

- www.agilemanifesto.org



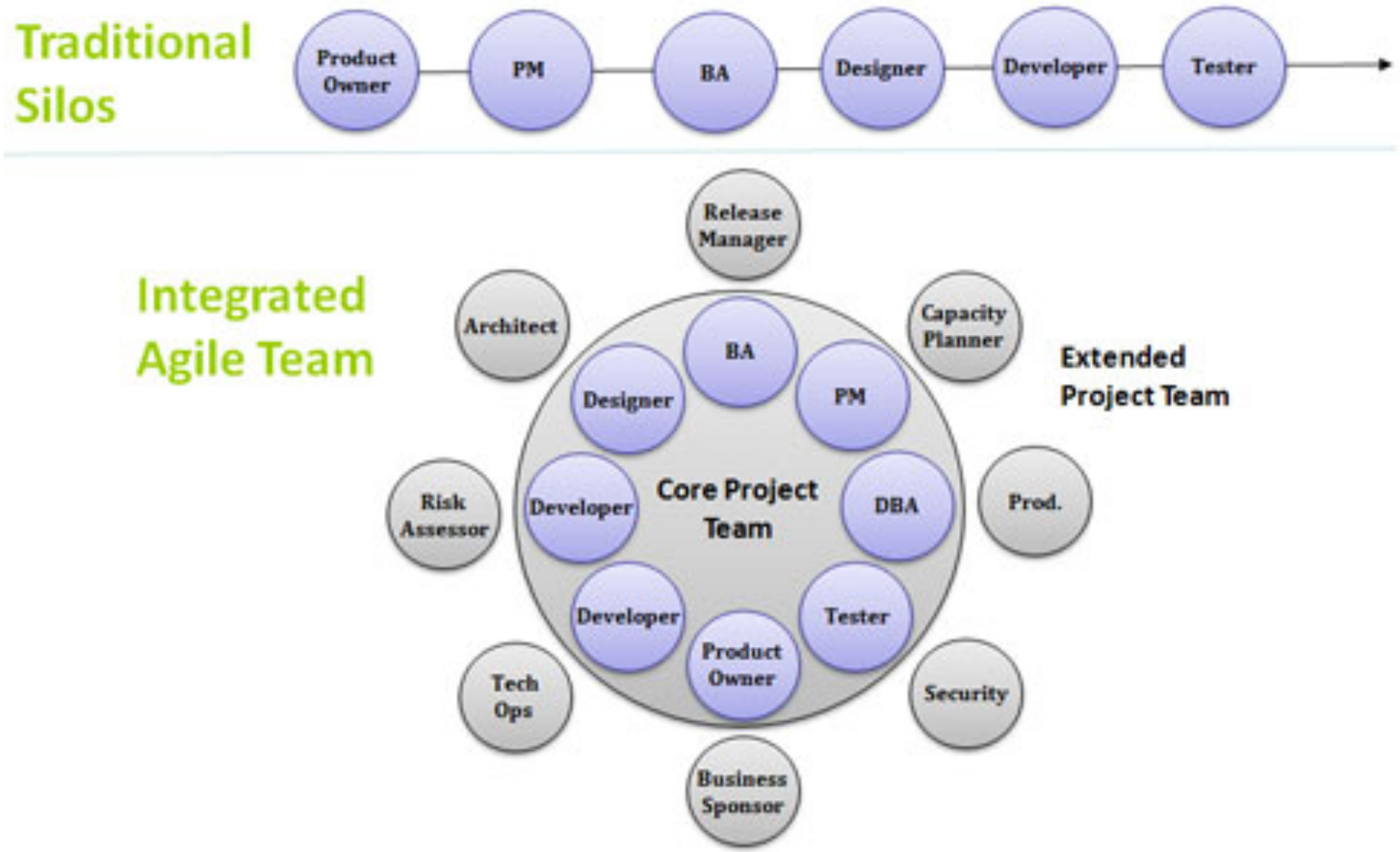
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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 prefer the items on the left.

- Management can tend to prefer the things on the right over the things on the left

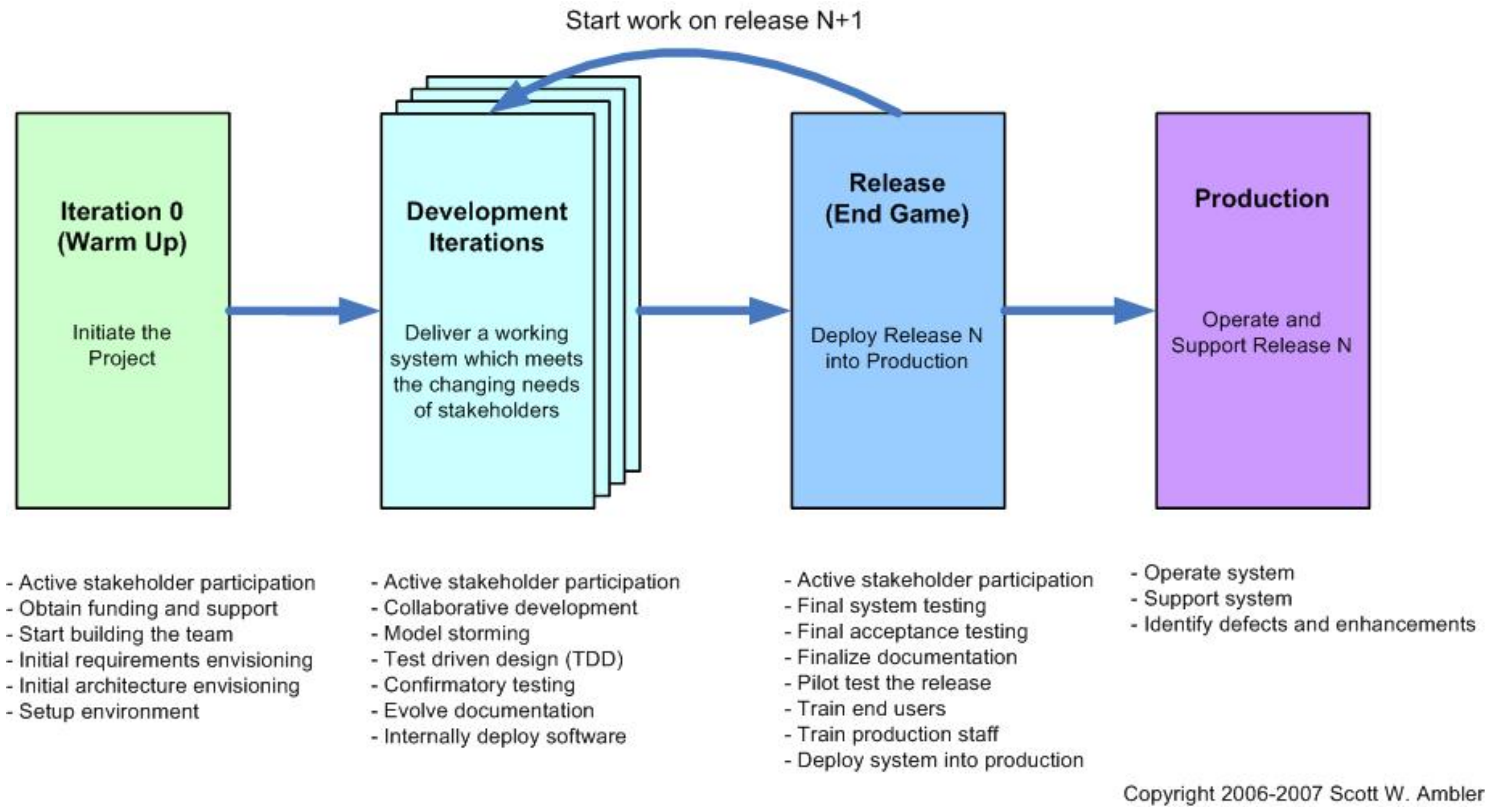
Traditional vs agile team



Agile development

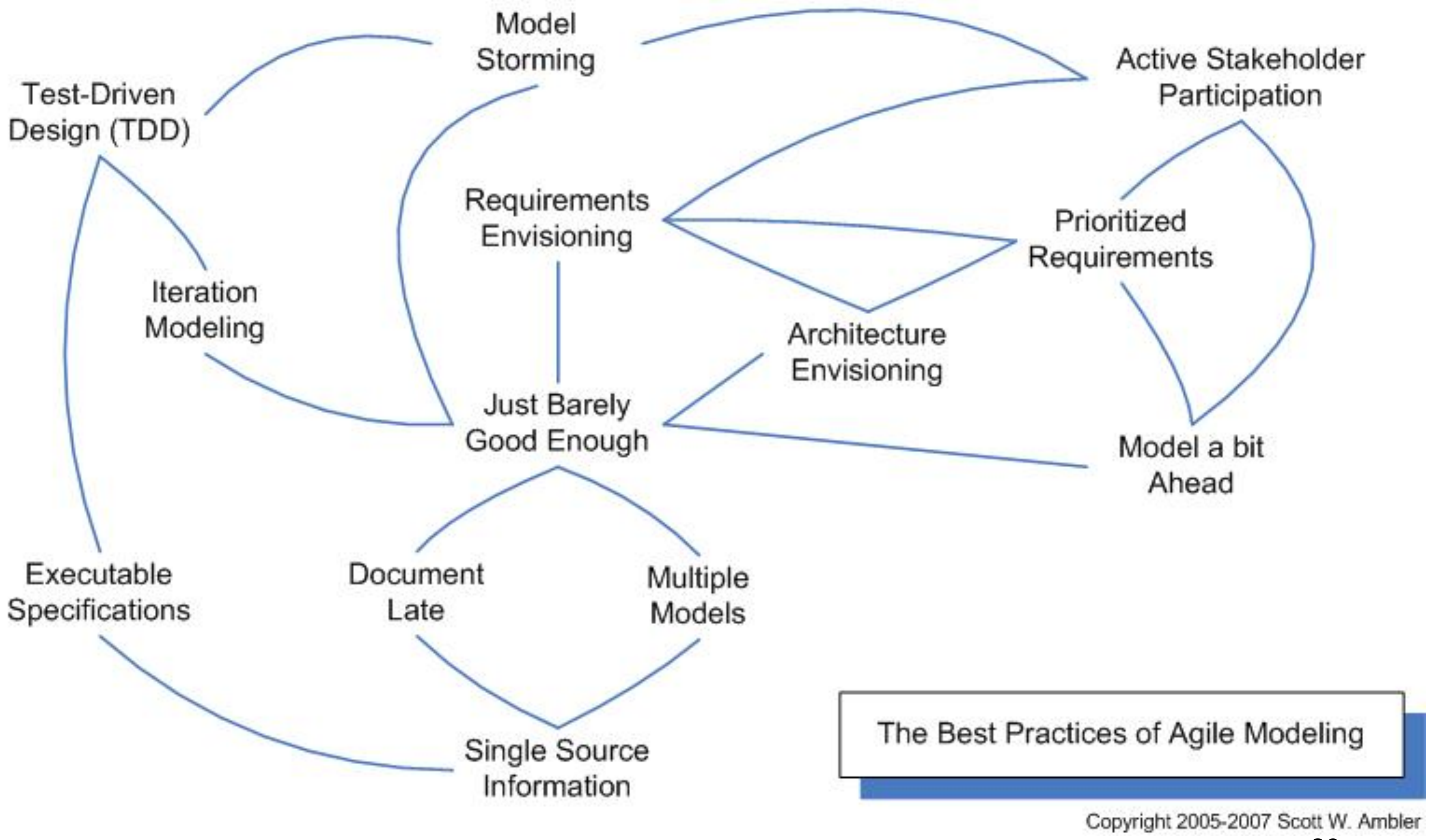
- Agile development uses feedback to make constant adjustments in a highly collaborative environment
- There are many agile development methods; most minimize risk by developing software in short amounts of time
- Software developed during one unit of time is referred to as an *iteration*, which typically lasts from hours or few days
- Each iteration passes through a full software development cycle

The Generic Agile Lifecycle

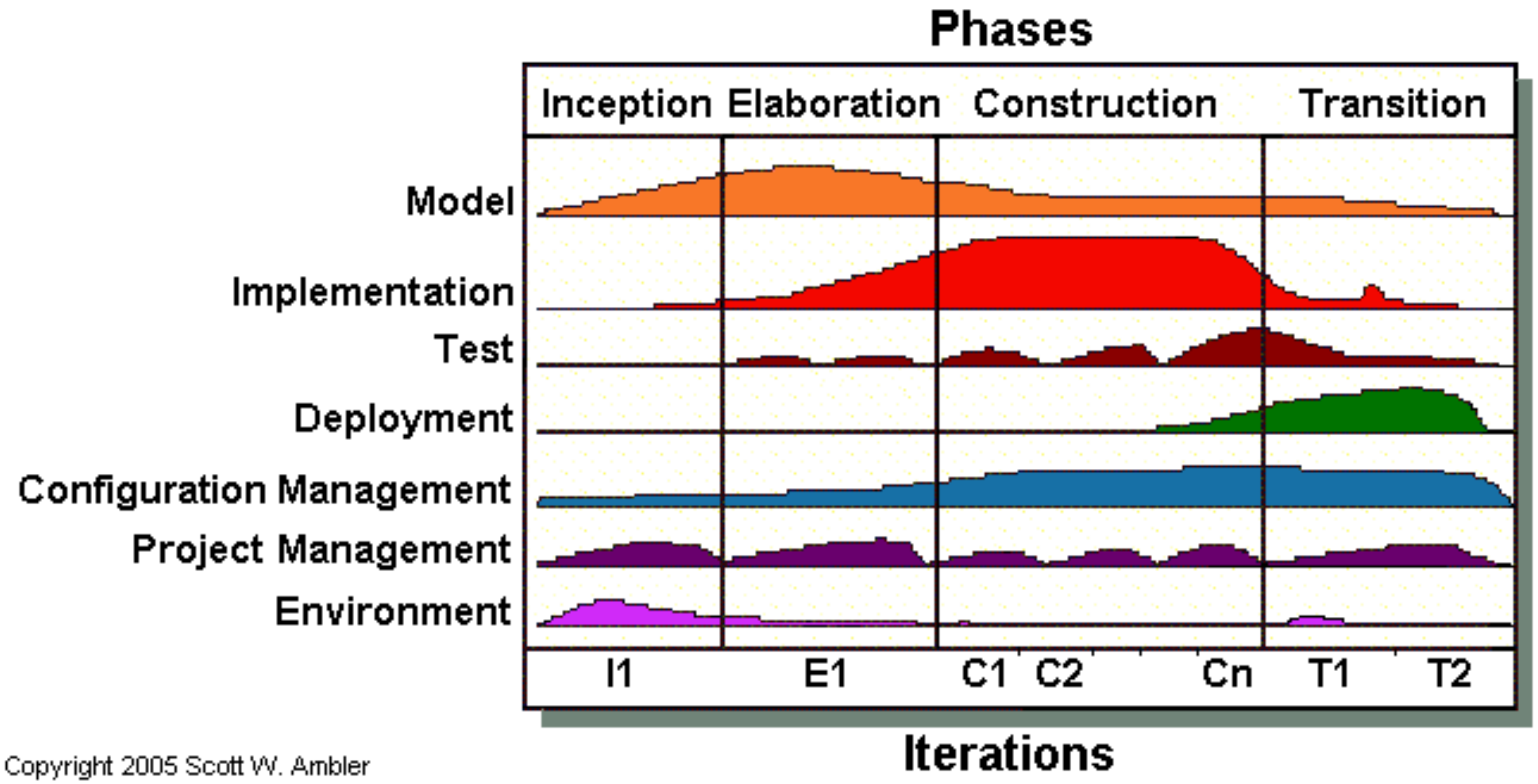


Start work on release N+1
Iteration 0 (Warm Up)
Initiate the Project
Development Iterations
Deliver a working system which meets the changing needs of stakeholders
Release (End Game)
Deploy Release N into Production
Production
Operate and Support Release N
- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements envisioning
- Initial architecture envisioning
- Setup environment
- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Confirmatory testing
- Evolve documentation
- Internally deploy software
- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production
- Operate system
- Support system
- Identify defects and enhancements
Copyright 2006-2007 Scott W. Ambler

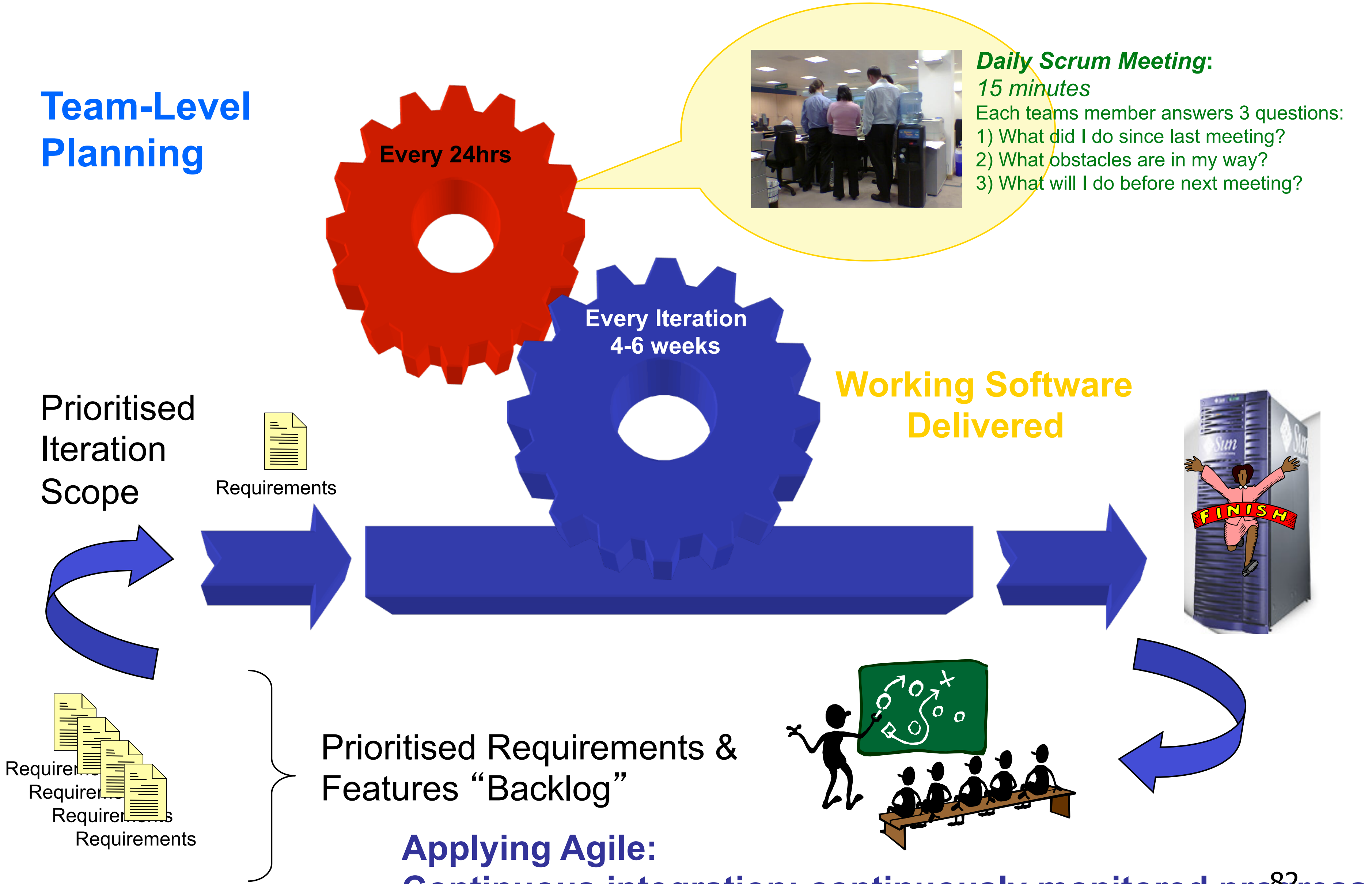
Agile practices



Agile Unified Process



SCRUM


Team-Level Planning
Every 24hrs
Every Iteration 4-6 weeks
Working Software Delivered
Prioritised Iteration Scope
Requirements
FINISH
Requirements
Requirements
Requirements
Requirements
Prioritised Requirements & Features "Backlog"

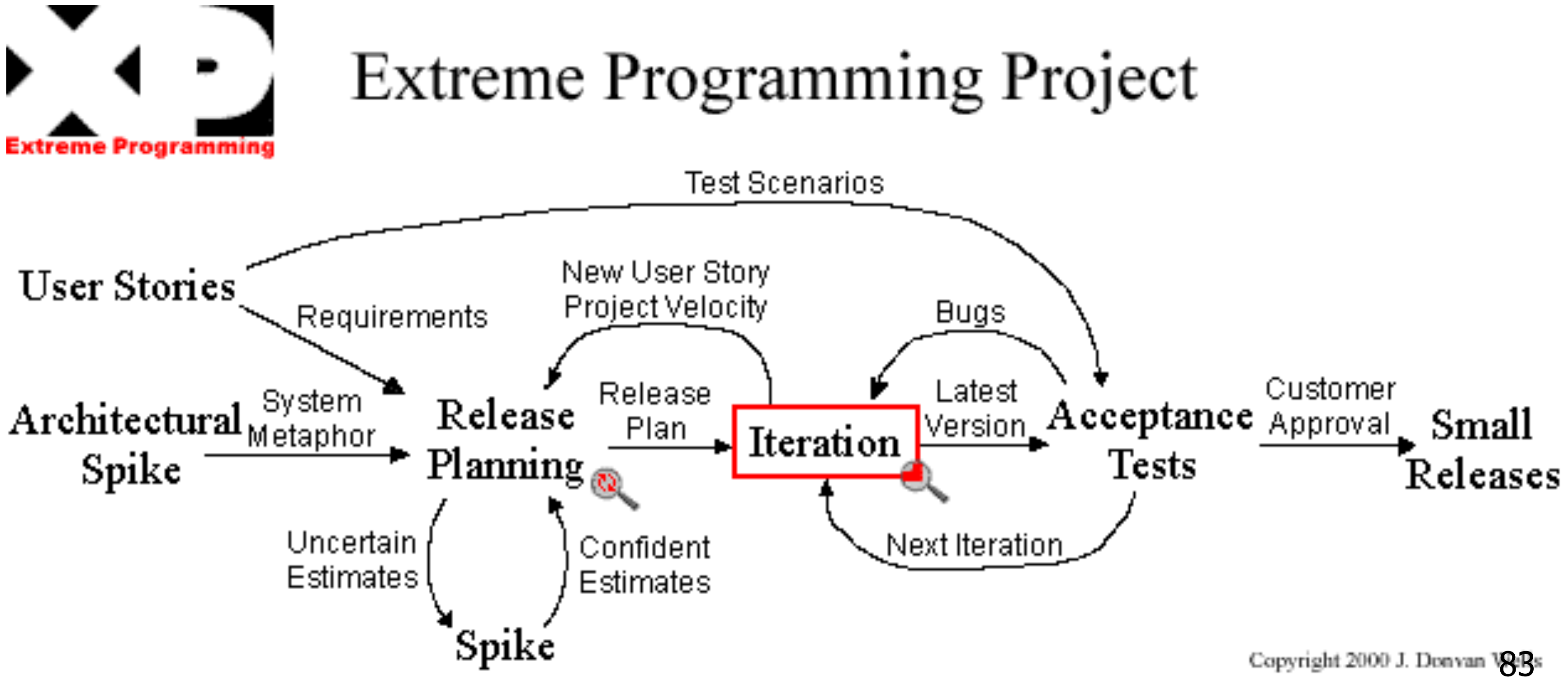
Daily Scrum Meeting:
15 minutes
Each teams member answers 3 questions:
1) What did I do since last meeting?
2) What obstacles are in my way?
3) What will I do before next meeting?

Applying Agile:
Continuous integration; continuously monitored progress

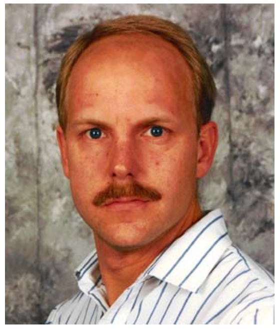
Agile: eXtreme Programming

The ethic values of eXtreme Programming

- Communication
- Simplicity
- Feedback
- Courage
- Respect (added in the latest version)



eXtreme Programming (XP)



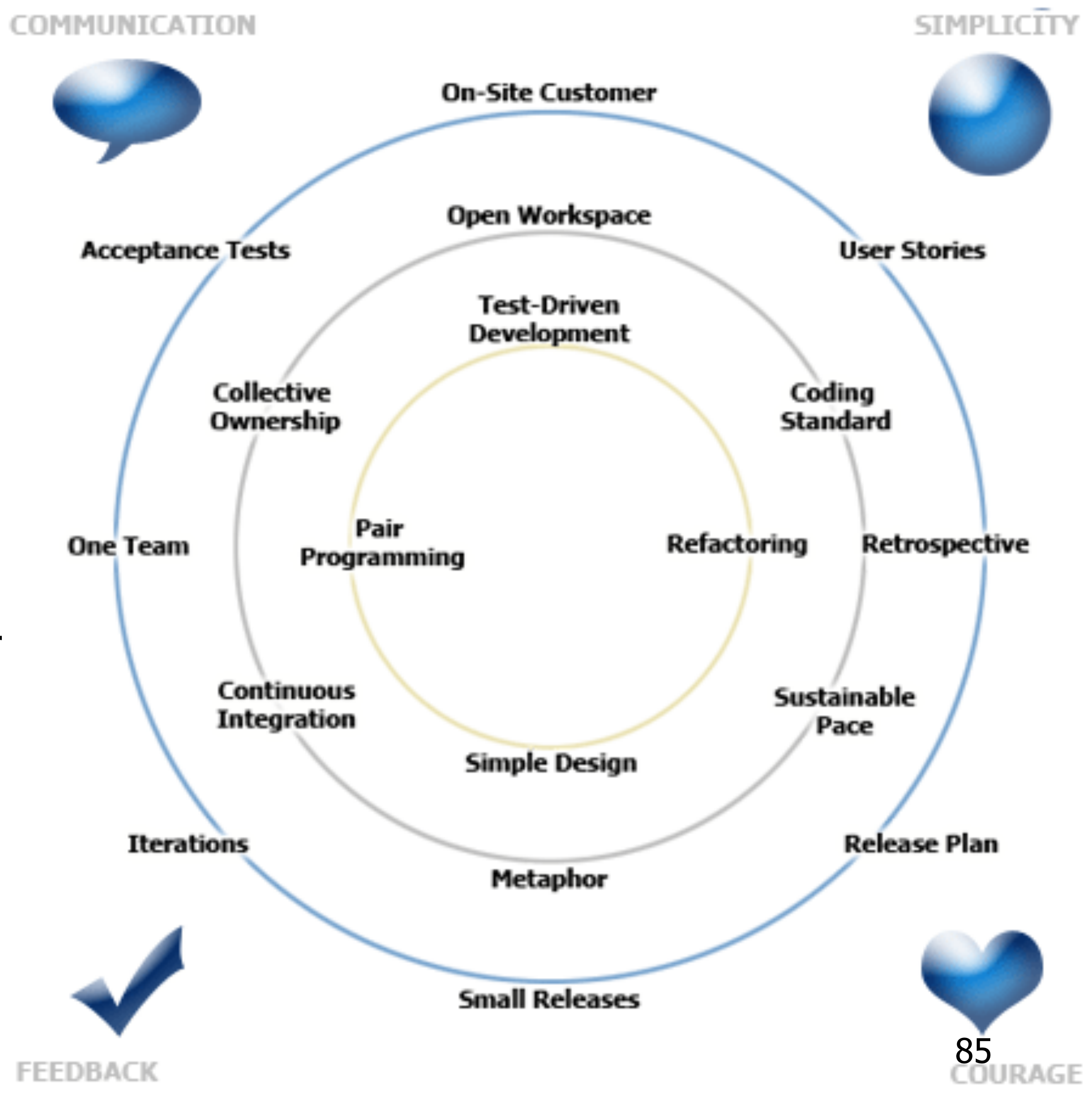
Kent Beck

- "Extreme Programming is a discipline of software development based on values of *simplicity*, *communication*, *feedback*, and *courage*"
- Proponents of XP and agile methodologies regard ongoing changes to requirements as a natural and desirable aspect of sw projects
- They believe that adaptability to changing requirements at any point during the lifecycle is a better approach than attempting to define all requiremenmts at the beginning and then expending effort to control changes to the requirements

`www.extremeprogramming.org`

The 12 Practices of XP

1. Metaphor
2. Release Planning
3. Testing
4. Pair Programming
5. Refactoring
6. Simple Design
7. Collective Code Ownership
8. Continuous Integratior
9. On-site Customer
10. Small Releases
11. 40-Hour Work Week
12. Coding Standards



Conclusions

Software engineering deals with

- the way in which software is made (process),
- the languages to model and implement software,
- the tools that are used, and
- the quality of the result (testing)

Self test questions

- How does Software Engineering differ from programming?
- Why is the “waterfall” model unrealistic?
- What is the difference between analysis and design?
- Why plan to iterate? Why develop incrementally?
- Why is programming only a small part of the cost of a “real” software project?
- What are the key advantages and disadvantages of object-oriented methods?

References: books

- Pressman, *Software engineering a practictioner approach*, 7th ed., McGrawHill, 2009
- Ambler, *Agile Modern Driven Development with UML2* (The Object Primer 3ed.), Cambridge Univ. Press, 2004
- Larman, *Agile and Iterative Development: a managers' guide*, Addison Wesley, 2003
- The Computer Society, *Guide to the Software Engineering Body of Knowledge*, 2013
  www.computer.org/portal/web/swebok

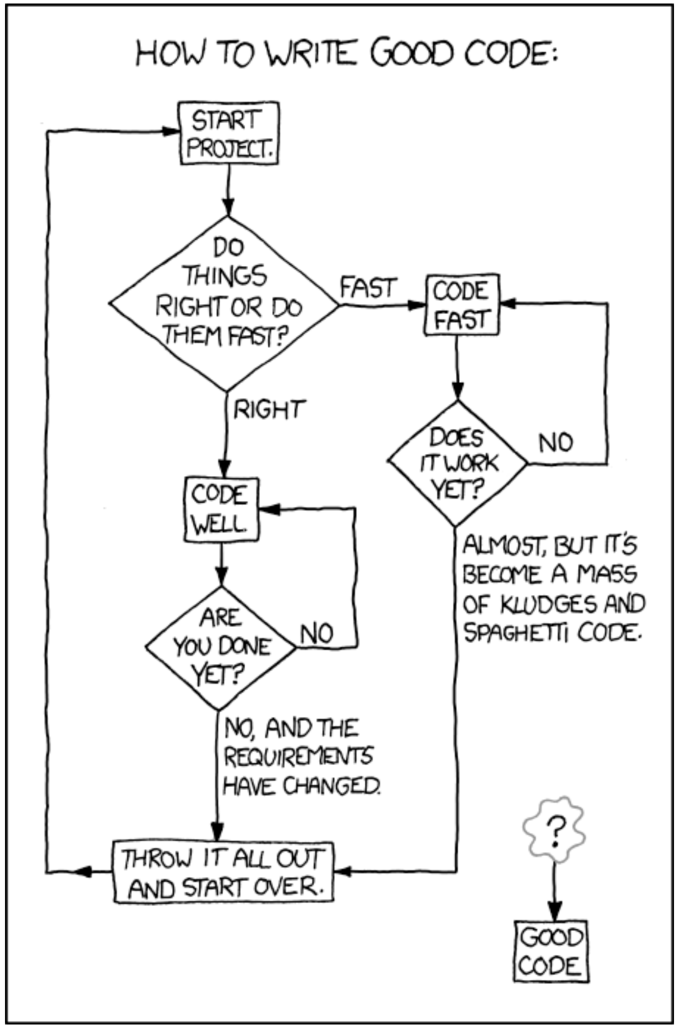
Reference: papers

- V.Basili et al., Understanding the High-Performance- Computing Community: A Software Engineer's Perspective, IEEE Software, 2008
- D.Lugato et al., Model-driven engineering for HPC applications, *Proc. Modeling Simulation and Optimization Focus on Applications, Acta Press* (2010): 303-308.
- M.Palyart et al, MDE4HPC: An Approach for Using Model-Driven Engineering in High-Performance Computing, Proc. SDL, LNCS 7083, 2011.

Useful sites

- `software-carpentry.org` Software carpentry
- `software.ac.uk/resources/case-studies`
- First Int. Workshop on Sw eng for HPC, 2013
  `sehpccse13.cs.ua.edu`

Questions?


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

http://xkcd.com/844/