The role of architecture in the software lifecycle

Prof. Paolo Ciancarini
Corso di Architettura del Software
CdL M Informatica
Università di Bologna

Agenda

- What is a software development process and how it is described
- The role of architecture in the software development process

Software: the product of a process

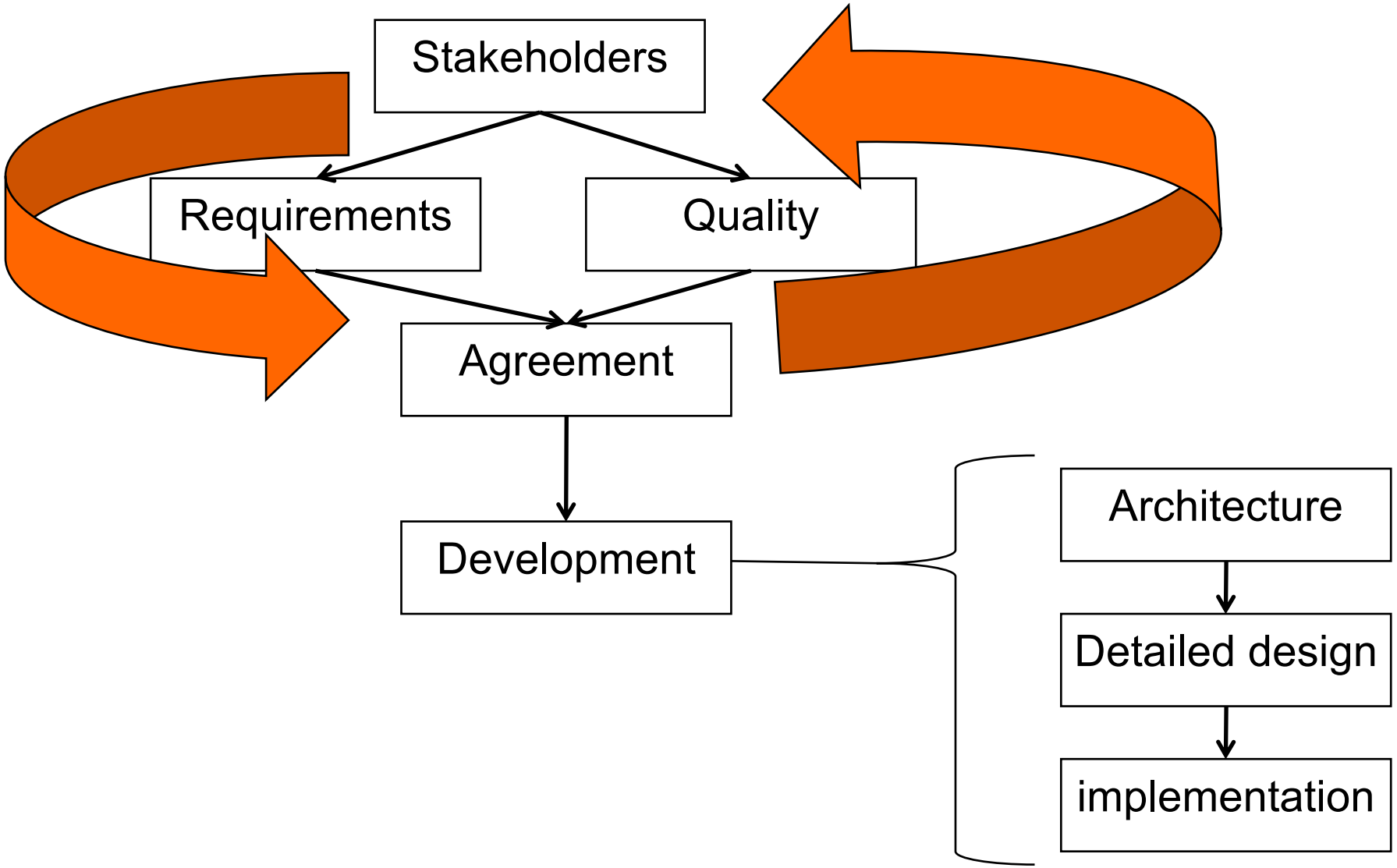
- Many kinds of software products → many kinds of development processes
- The development process has the goal of satisfying “most” stakeholders’ concerns
- Improve the process to improve the product

- Examples of process models: waterfall, iterative, agile, model-driven, …

Lifecycles

- A lifecycle is a process for creating or using a system or a product
- Systems/products and their lifecycles can be evaluated for quality
- Software systems/products result from several entangled lifecycles:
  - Industrial lifecycle
  - Development lifecycle (eg.: reqs, build, test)
  - Operational lifecycle (eg.: portal with CMS)
  - Reengineering lifecycle (eg. legacy system)

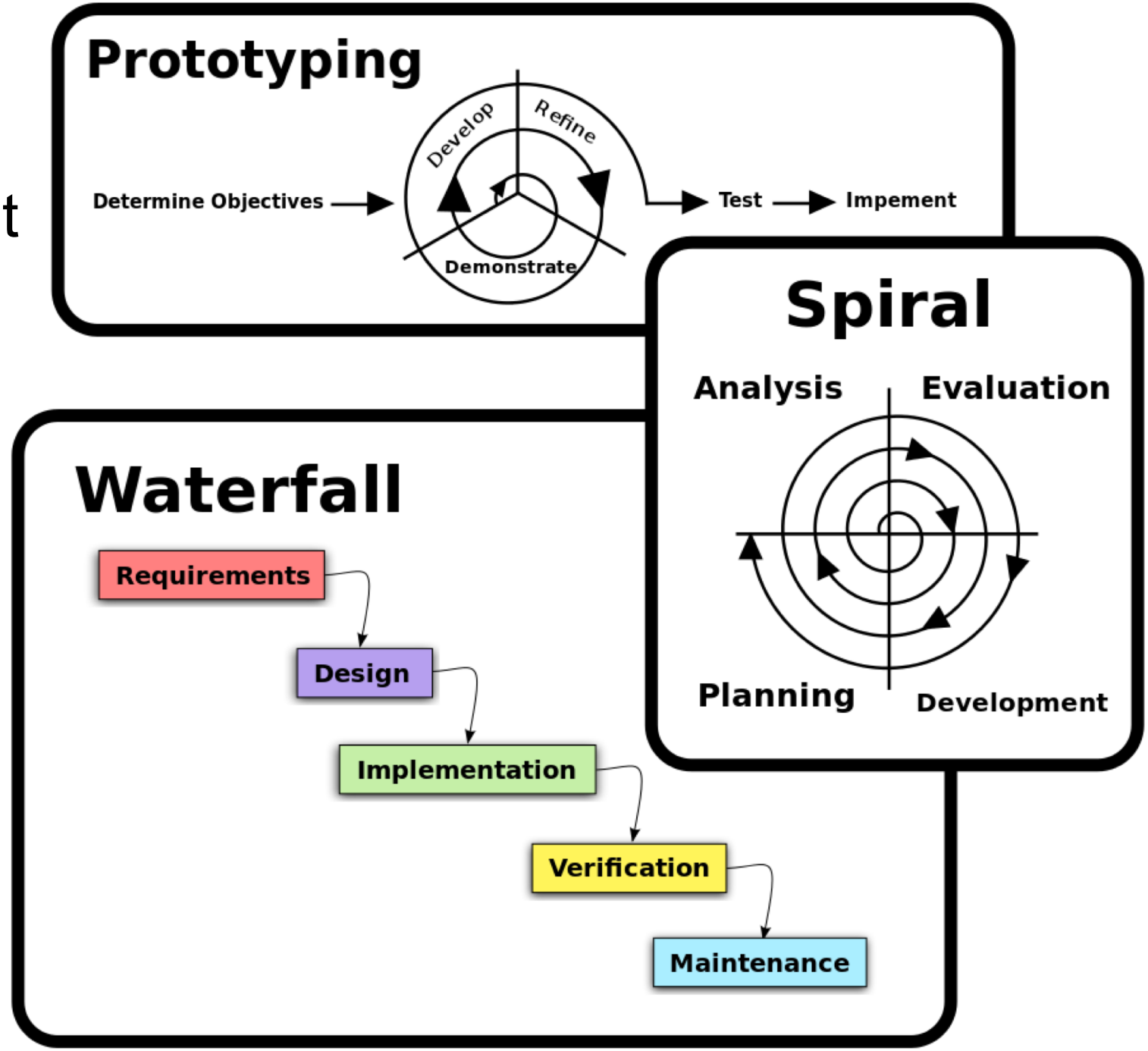
The lifecycle, traditional



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.

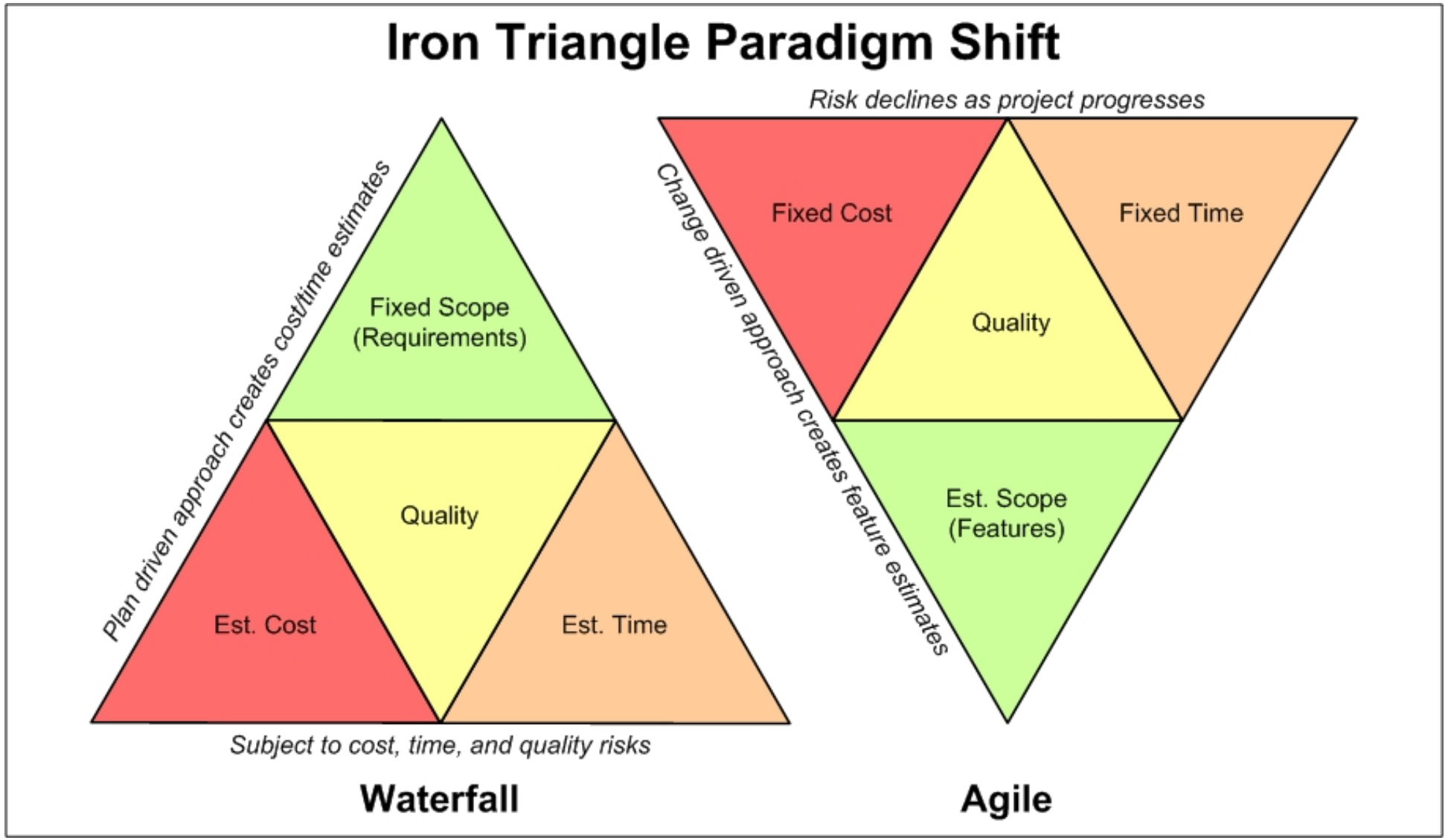
Sw development processes: examples

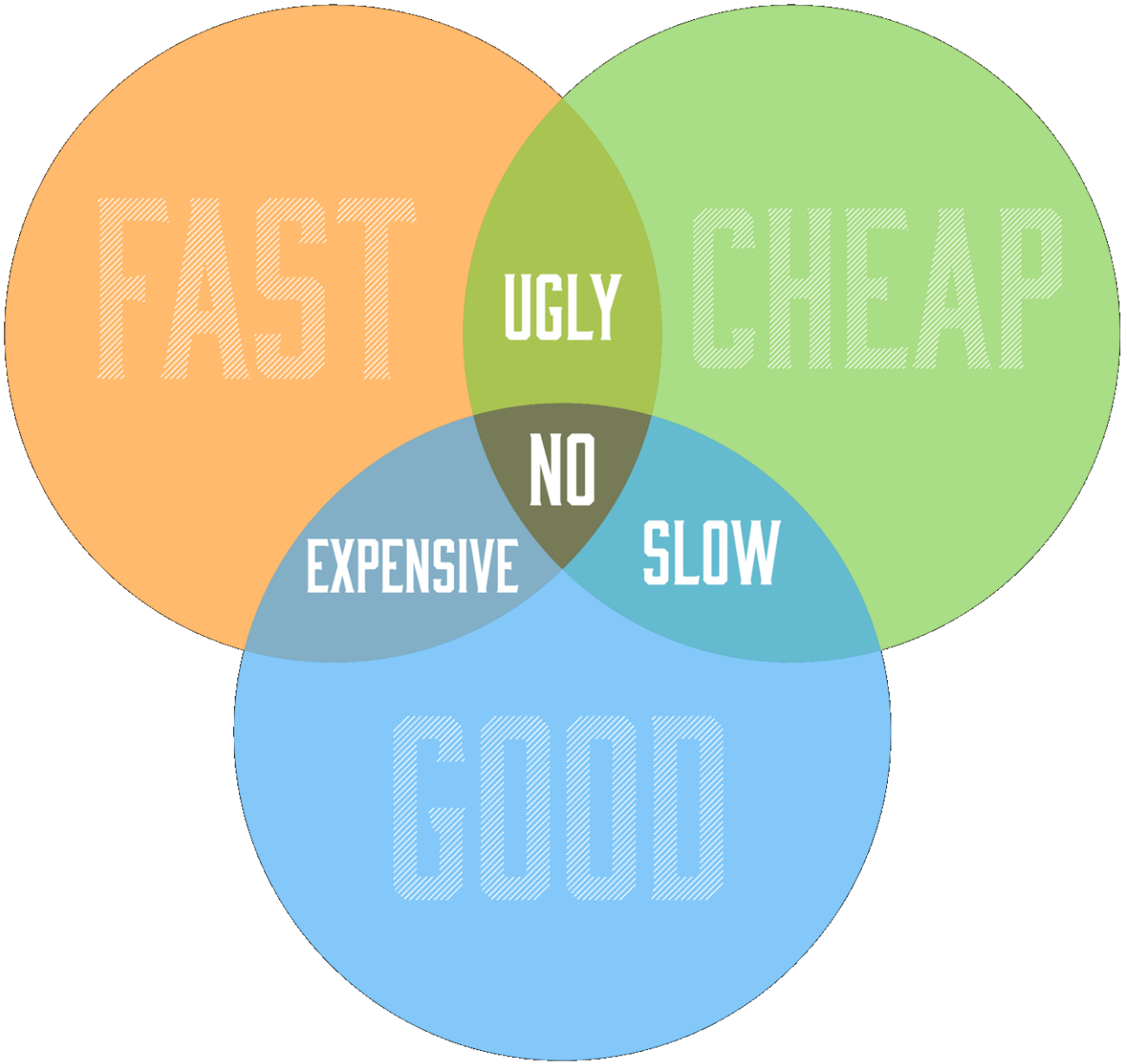


Control variables of sw processes

- 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

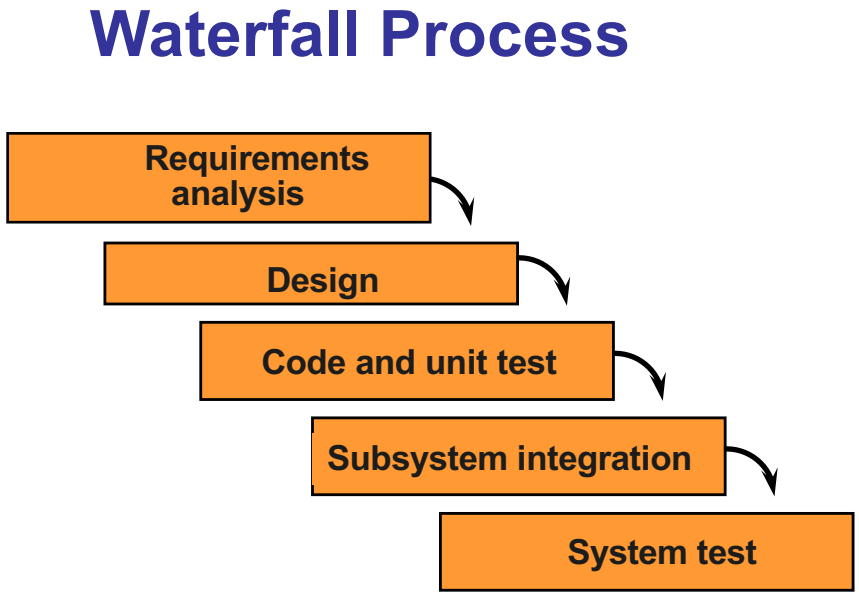

Iron Triangle Paradigm Shift
Plan driven approach creates cost/time estimates
Fixed Scope
(Requirements)
Quality
Est. Cost
Est. Time
Subject to cost, time, and quality risks
Waterfall
Risk declines as project progresses
Change driven approach creates feature estimates
Fixed Cost
Fixed Time
Quality
Est. Scope
(Features)
Agile


FAST
CHEAP
GOOD
UGLY
NO
EXPENSIVE
SLOW

Models for the software process

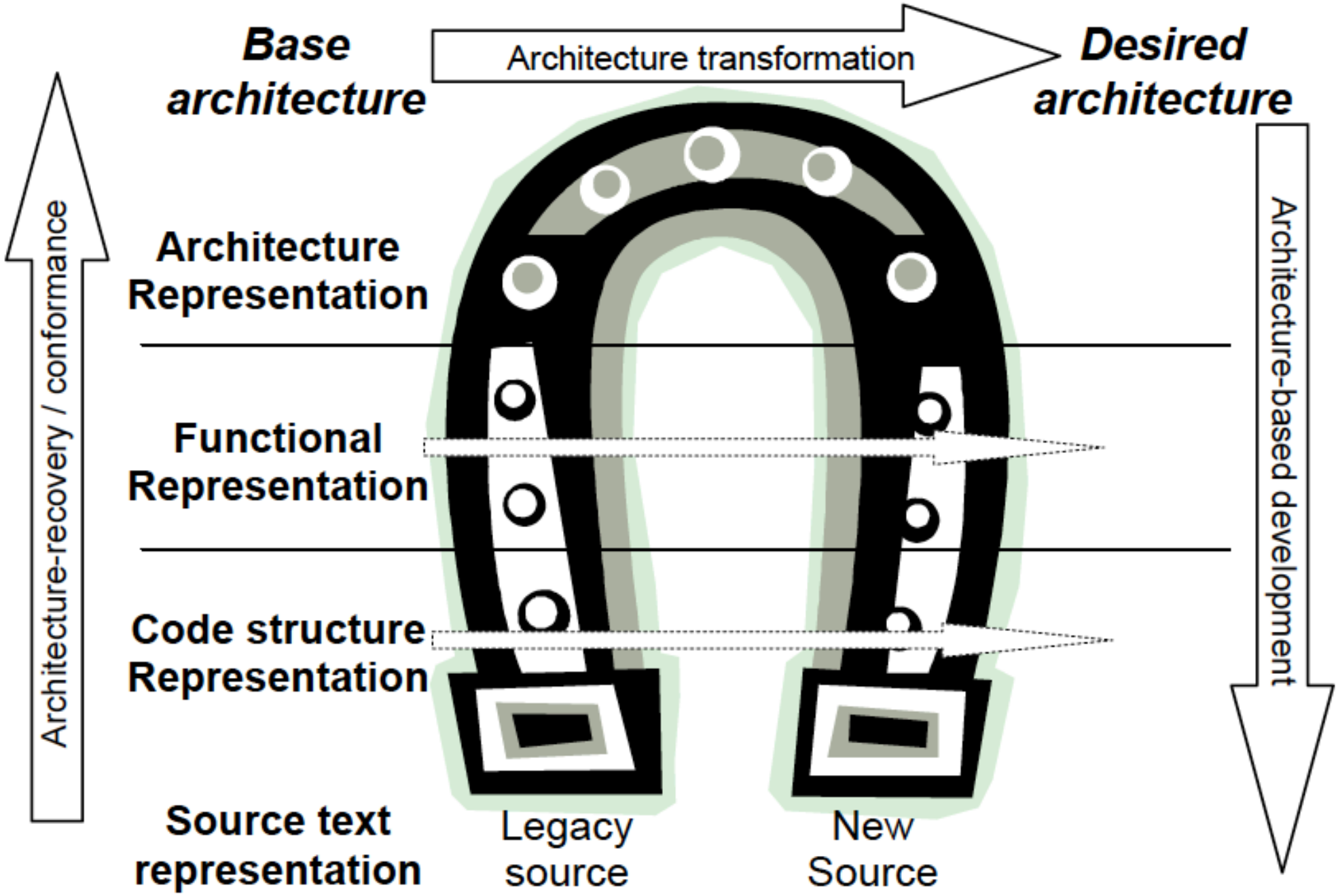
- Waterfall (planned, linear)
- Spiral (planned, iterative)
- Agile (unplanned, test driven)

Waterfall development

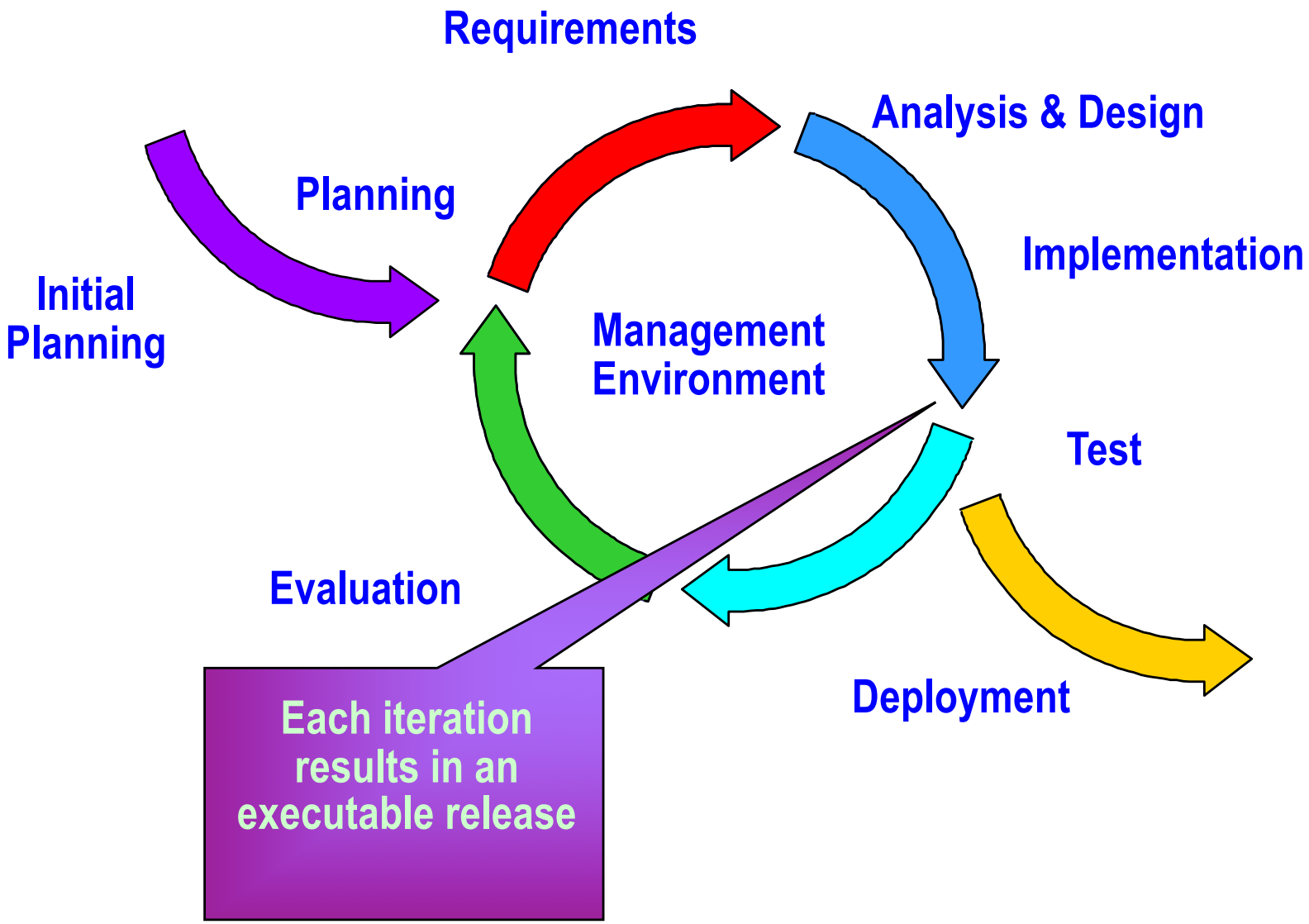


- Delays risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and mixes integration and testing
- Precludes early deployment
- Frequently: major unplanned iterations

Horseshoe: reengineering process for legacy sw



Iterative Development



Benefits of Iterative Development

A software project evolves in iterations to:

- Mitigate risk, because each iteration includes user validation
- Accommodate change of requirements
- Learn along the way
- Improve the quality of the artifacts
- Exploit reuse, thus increasing productivity

Problems with iterative processes

- How iterative? How many rounds?
- Flexible (agile) or rigid?
- Heavy weight (many rules, practices and documents) vs. lightweight (few rules and practices)
- Disciplined vs ad hoc (or chaotic)

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."

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

Agile developments methods

Some agile sw development methods

- Dynamic System Development Methodology and RAD (www.dsdm.org, 1995)
- Scrum (Sutherland and Schwaber, 1995)
- XP - eXtreme Programming (Beck, 1999)
- Feature Driven Development (DeLuca, 1999)
- Adaptive Sw Development (Highsmith, 2000)
- Lean Development (Poppendieck, 2003)
- Crystal Clear (Cockburn, 2004)
- Agile Unified Process (Ambler, 2005)

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.

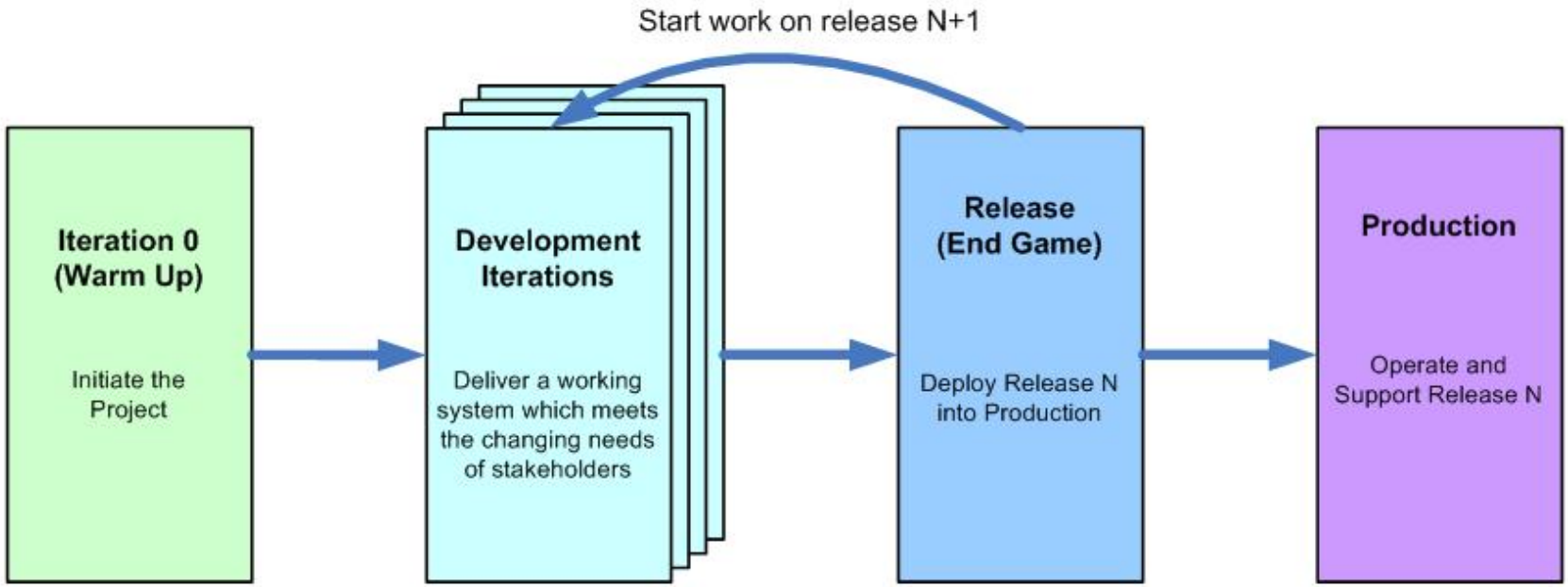
- Management can tend to prefer the things on the right over the things on the left

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



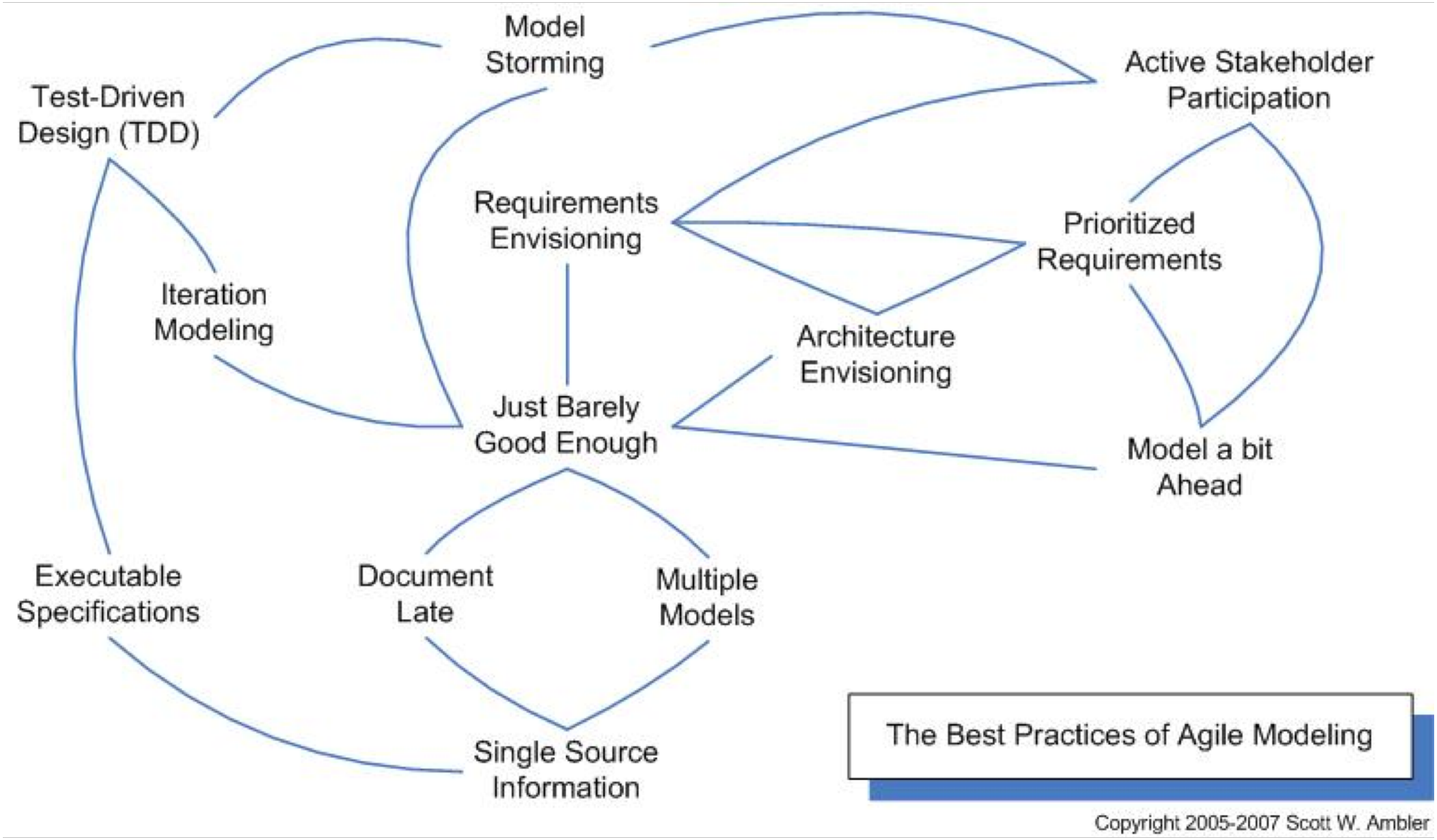
- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements envisioning
- Initial architecture envisioning
- Setup environment

- 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

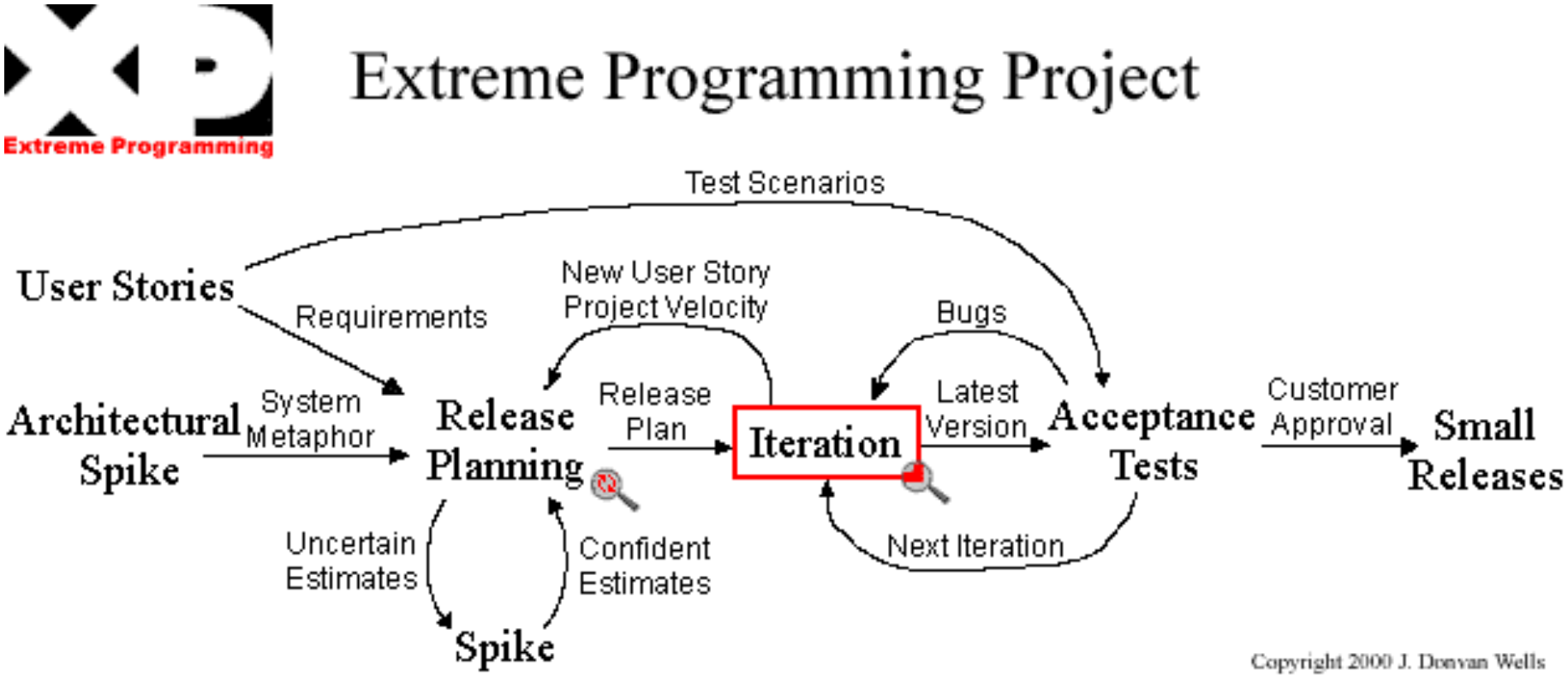
Agile practices



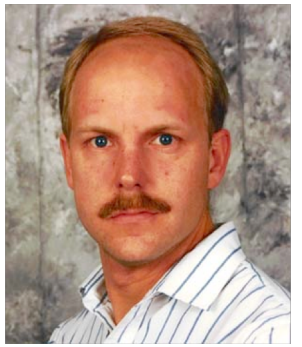
Agile: eXtreme Programming

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*, *courage, and respect*"
- 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 requirements at the beginning and then expending effort to control changes to the requirements

`www.extremeprogramming.org`

XP values

- **Communication**: simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication
- **Simplicity**: start with the simplest solution; extra functionality can then be added later
- **Feedback**: from the system (unit tests), the customer (acceptace tests), the team (estimations during the planning game)
- **Courage**: reviewing the existing system and modifying it so that future changes can be implemented more easily. Refactoring
- **Respect**: Programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers

SCRUM

Team-Level Planning

***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?

Every 24hrs

Every Iteration 4-6 weeks

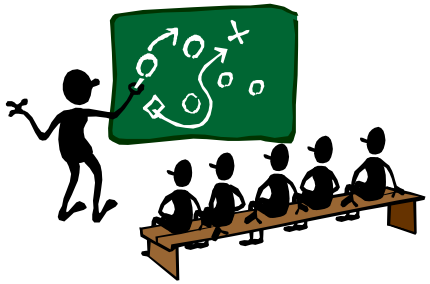
Prioritised Iteration Scope

Requirements

Working Software Delivered

Requirements

Prioritised Requirements & Features "Backlog"



Applying Agile:
Continuous integration; continuously monitored progress

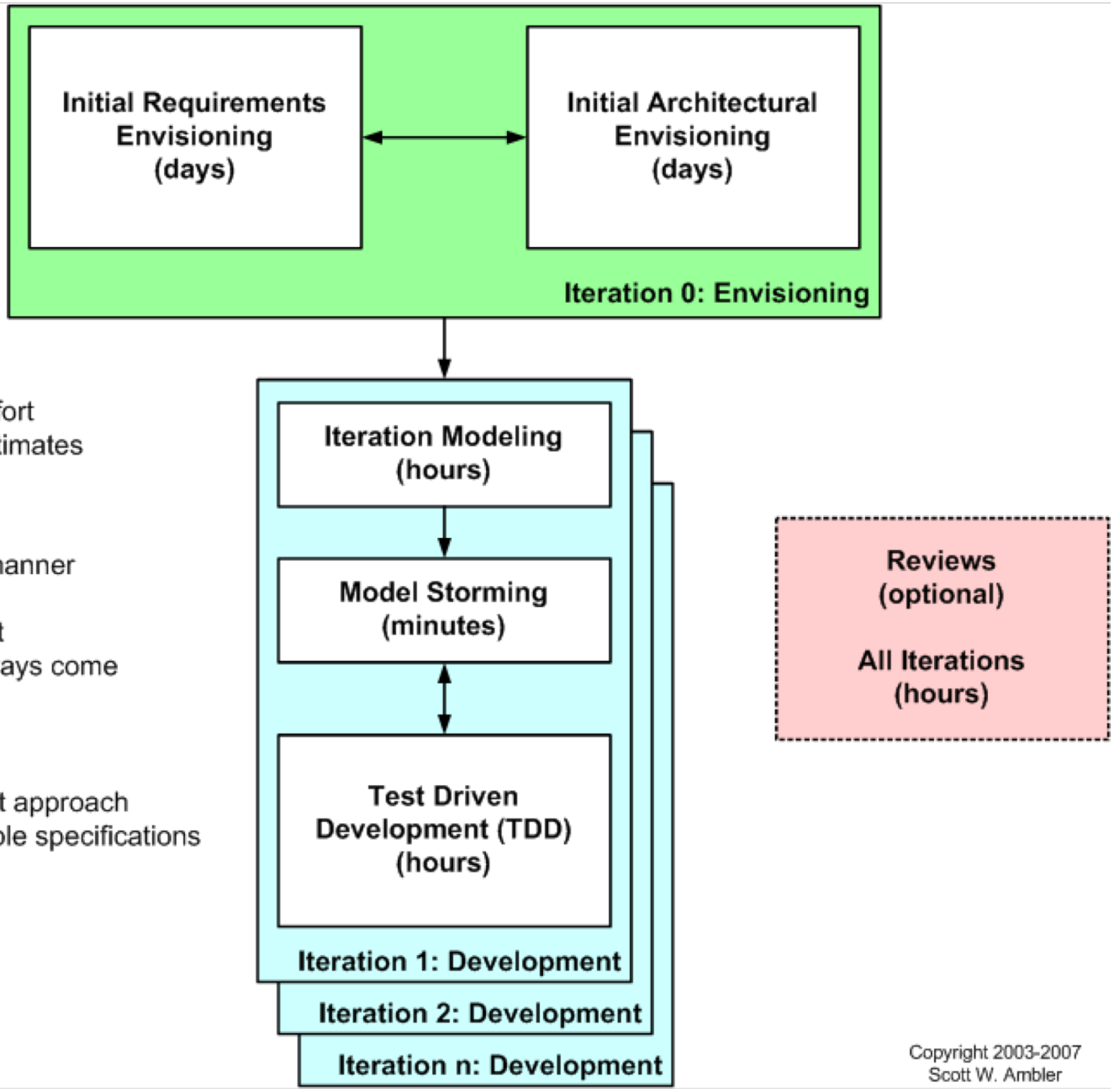
Agile development: architecture-centric

- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision

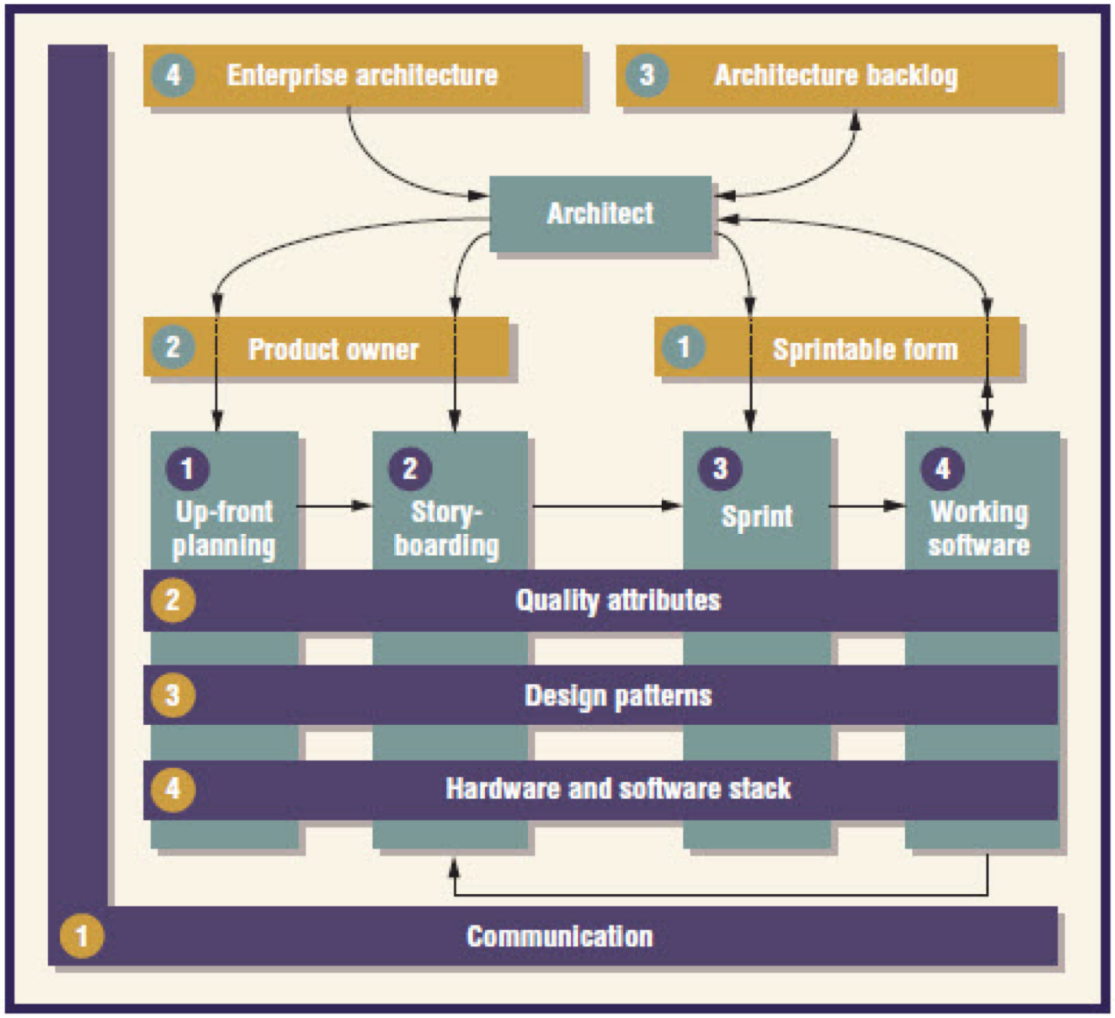
- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications



The agile architect



www.infoq.com/articles/agile-architecture

A framework for agile software architecture.

The architect's involvement during project execution helps to achieve the project objectives

Table 1

planation of the elements in a hybrid framework

Category	Item	Description
Interaction point	1. Up-front planning	Setting the architectural direction in much the same way as sequential projects
	2. Storyboarding	Structuring the business need and architectural work, and getting everyone on board
	3. Sprint	Building the functionality as part of the team when direct participation is valuable
	4. Working software	Reviewing what's actually delivered to measure the architectural state
Critical skill	1. Sprintable form	Breaking architectural work into small, measurable units
	2. Product owner	Quantifying the architecture in terms of clear business value
	3. Architecture backlog	Tracking architectural concerns and balancing them with business priorities
	4. Enterprise architecture	Knowing the larger architectural picture and using each project to advance it
Architectural function	1. Communication	Keeping all stakeholders informed about the architecture's value and state
	2. Quality attributes	Measuring maintainability, scalability, extensibility, and similar "-ilities"
	3. Design patterns	Outlining the structures that give form to implementation work
	4. Hardware and software stack	Choosing appropriate hardware and software for the project

Table 2

Applying architectural functions at agile interaction points

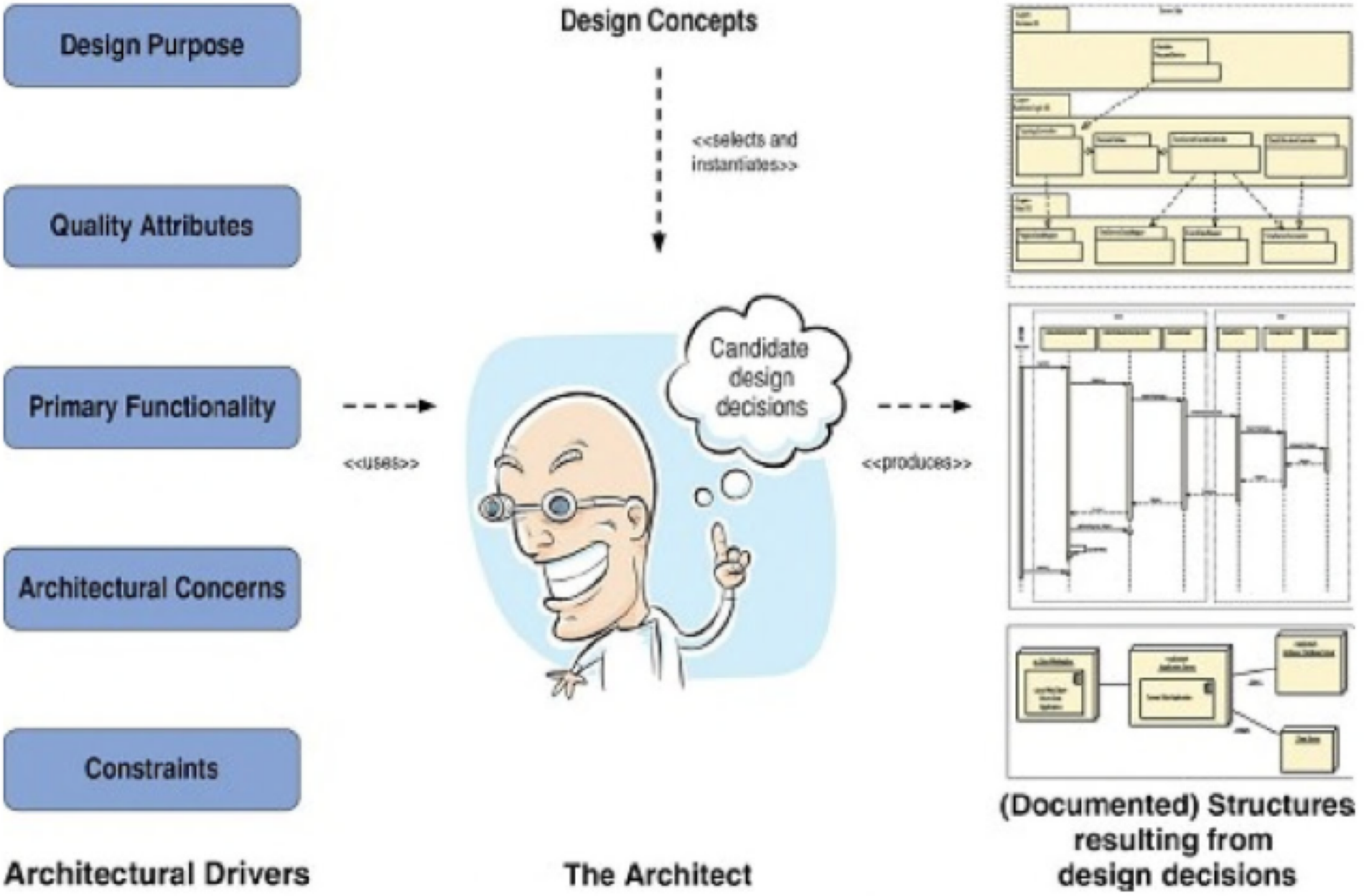
Architectural function	Interaction point			
	Up-front planning	**Storyboarding**	**Sprint**	**Working software**
Communication	■ Understand business objectives. ■ Get input from the technical team. ■ Communicate the general direction to everyone.	■ Actively facilitate storyboarding sessions. ■ Work architectural user stories into the backlog, particularly the types in the three cells immediately below:	■ Attend daily stand-ups. ■ Build functionality as a means of gaining understanding. ■ Mentor and assist as expertise allows.	■ Attend the sprint review. ■ Review documentation. ■ Advocate refactoring for architectural value with the team and product owner.
Quality attributes	■ Set approximate target ranges for attributes. ■ Establish which attributes dominate in trade-offs.	■ Add stories to improve specific attributes, including refactoring.	■ Build attributes into code, explicitly and as a norm for build work. ■ Assist in designing or building to improve attributes.	■ Verify that the delivered solution meets target ranges. ■ Adjust target ranges if build work indicates a need for adjustment.
Design patterns	■ Choose important design patterns. ■ Outline general interactions among significant patterns.	■ Add stories to build design patterns, including refactoring.	■ Solve for detailed design patterns. ■ Assist in building the most critical design patterns.	■ Verify that the delivered design patterns are valid. ■ Adjust design patterns as build work indicates.
Hardware and software stack	■ Reuse the corporate stack. ■ Prototype early to verify assumptions. ■ Plan carefully; hardware and software changes are inherently nonagile.	■ Add stories designed to validate hardware and software.	■ Validate hardware and software selection in early sprints. ■ Change early and quickly if stack needs adjusting.	■ Verify hardware and software by continually delivering business functionality on it. ■ Deploy to other environments routinely.

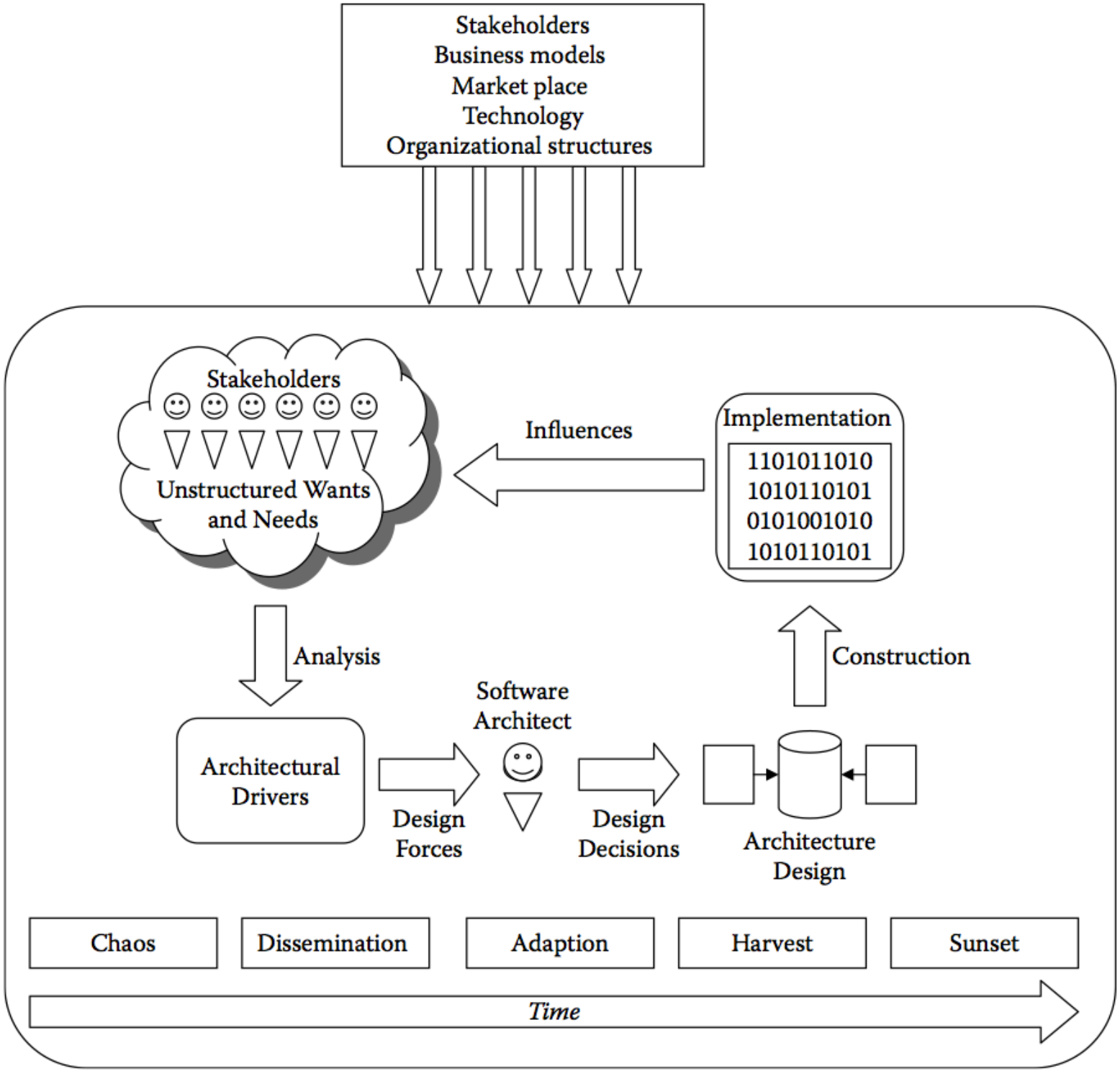
Architecture-centric processes

Architectural principles

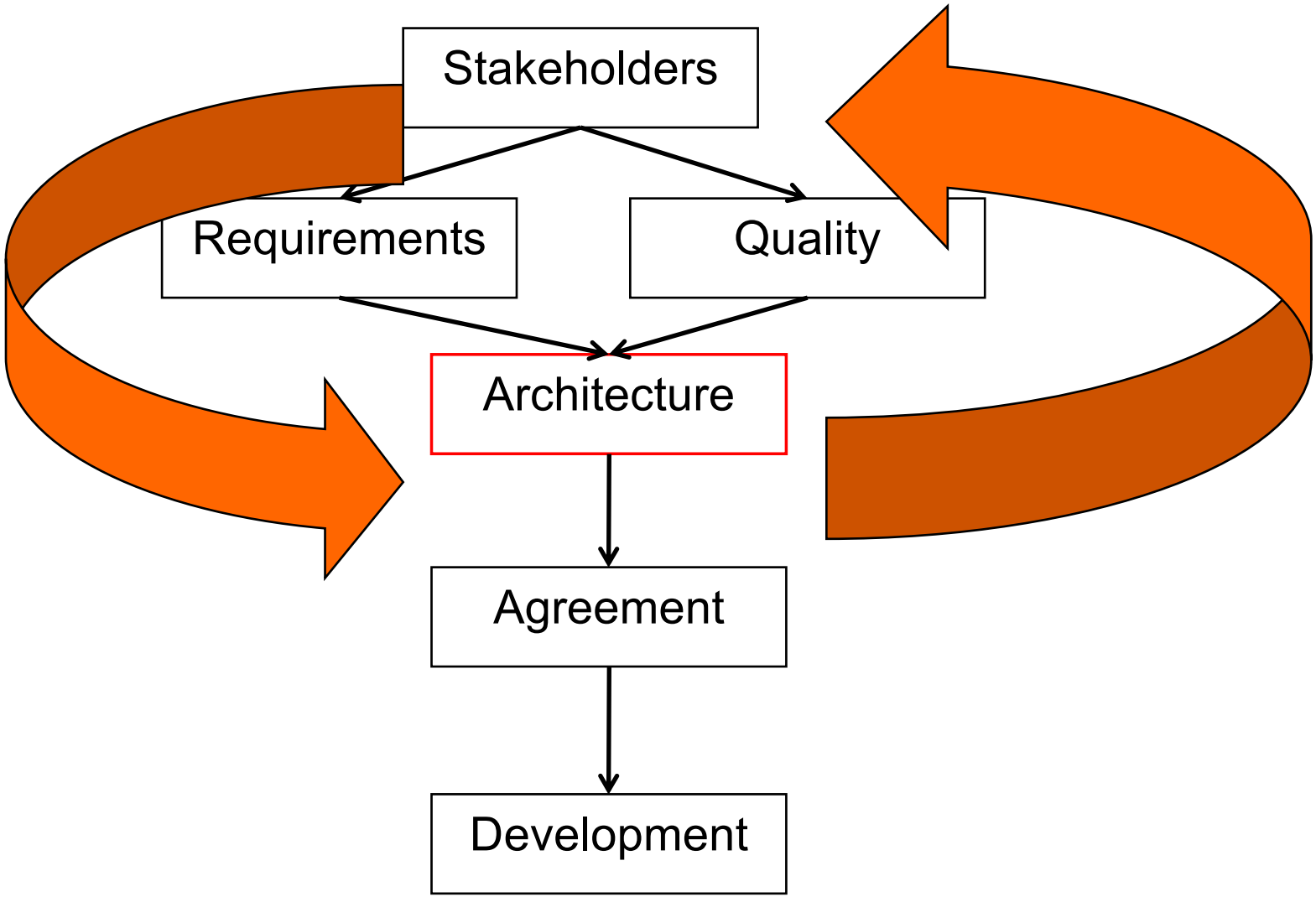
- Quality ("non-functional") requirements inspire the design of the software architecture
  - Quality attribute requirements stem from business goals
  - Key quality attributes need to be characterized in a system-specific way
  - Scenarios are a powerful way to characterize quality attributes and represent stakeholder views
- Sw architecture guides development throughout the life cycle
  - the architecture is central to all development activities
  - The development activities must have an explicit focus on quality attributes
  - The development activities must directly involve stakeholders, not just the architects

The inputs for architecting

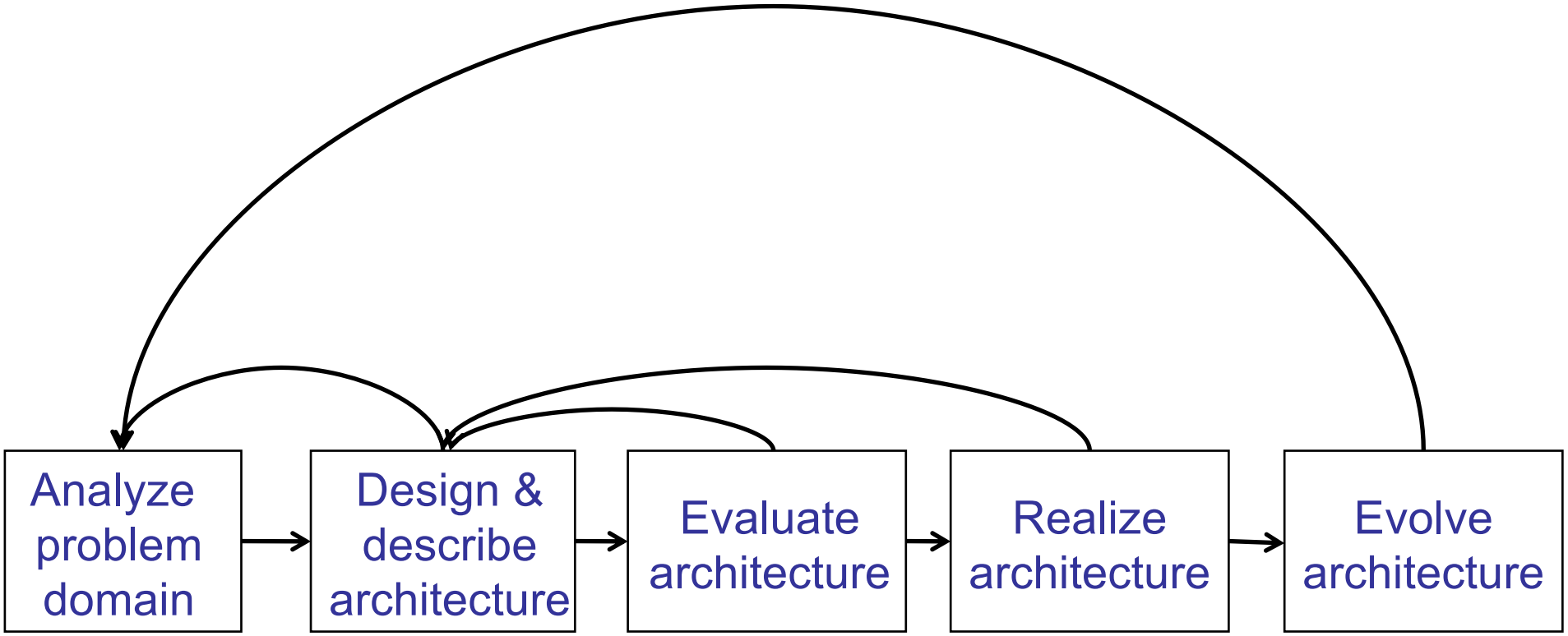



Stakeholders
Business models
Market place
Technology
Organizational structures
Stakeholders
Unstructured Wants and Needs
Influences
Implementation
1101011010
1010110101
0101001010
1010110101
Analysis
Construction
Software Architect
Architectural Drivers
Design Forces
Design Decisions
Architecture Design
Chaos
Dissemination
Adaption
Harvest
Sunset
Time

The lifecycle, architecture-centric

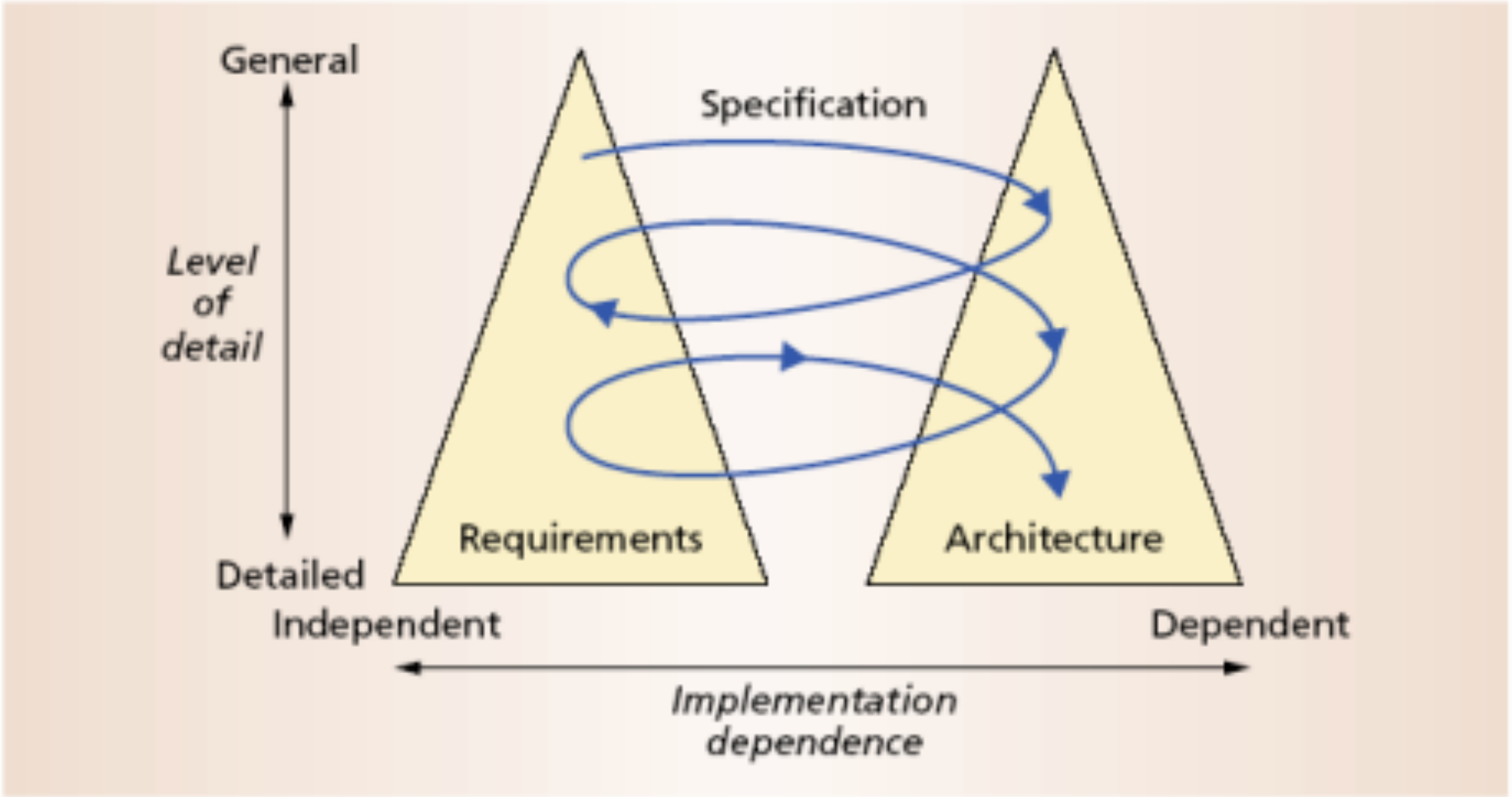


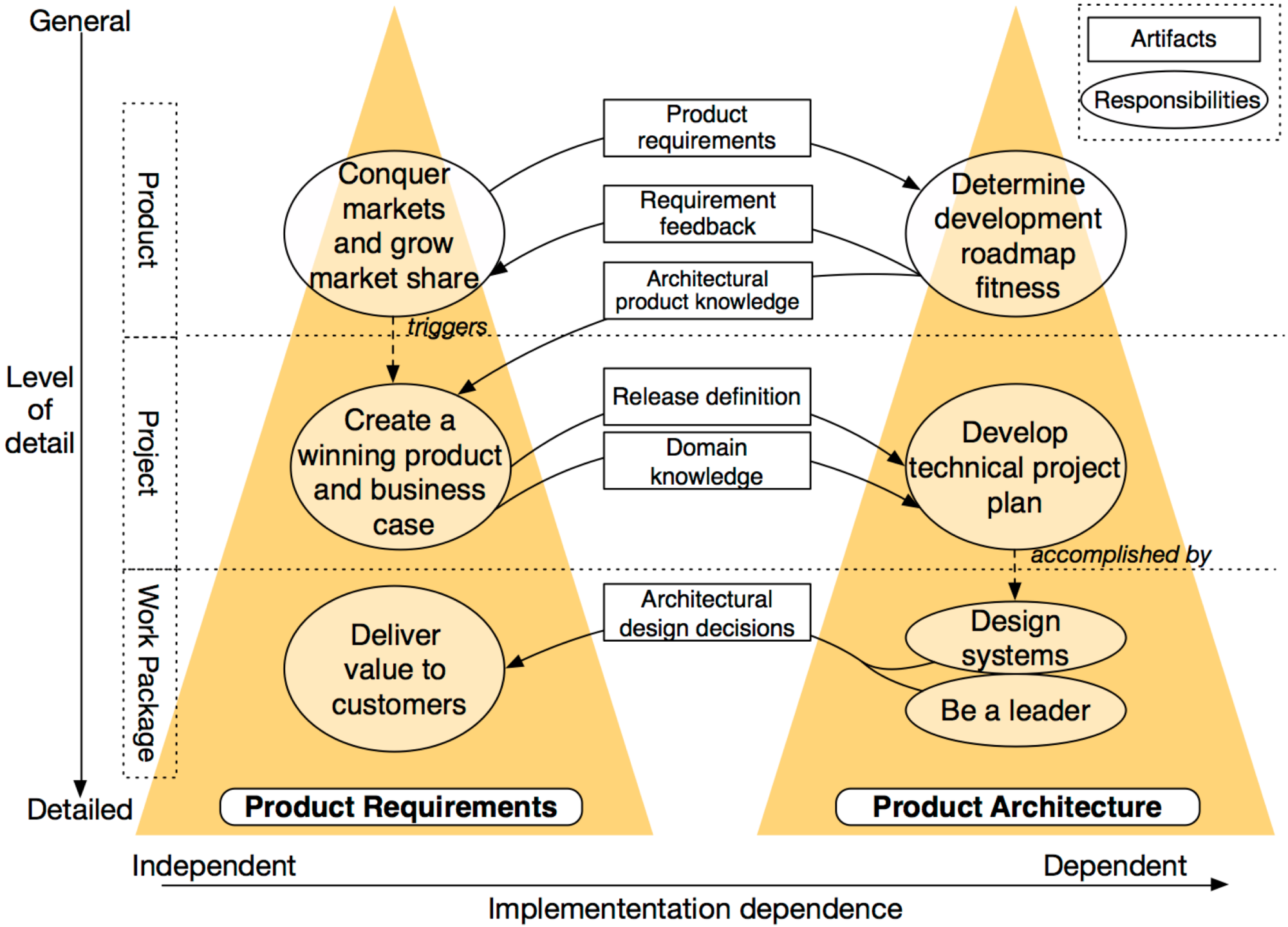
A model of the sw architecture process



Twin peaks process model

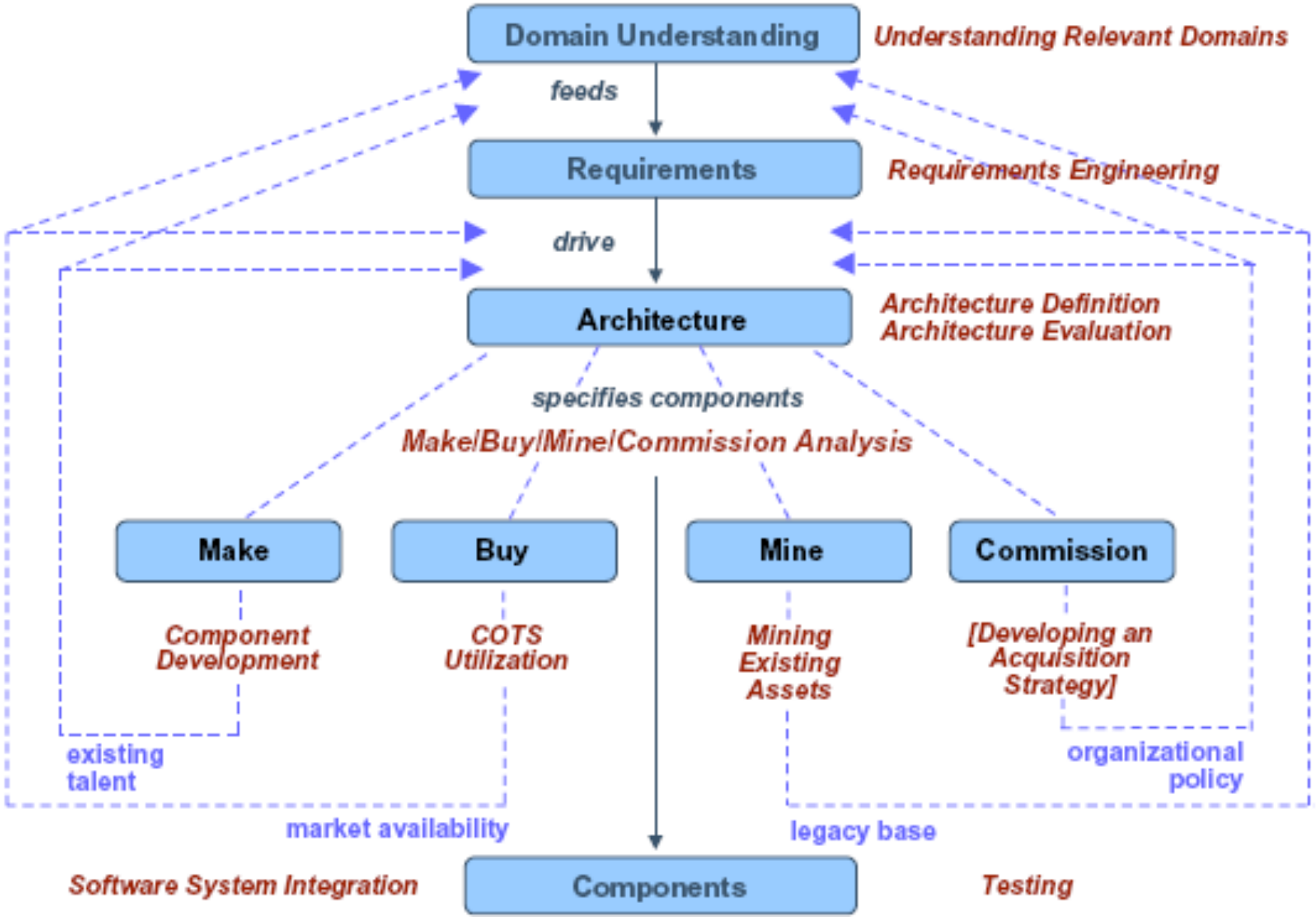
The early phases of a development process can be described by the *twin peaks process model*, where the specification of requirements and architecture proceed together





community.ispma.org/the-reciprocal-twin-peaks-lucassen-dalpiaz-werf-and-brinkkemper/

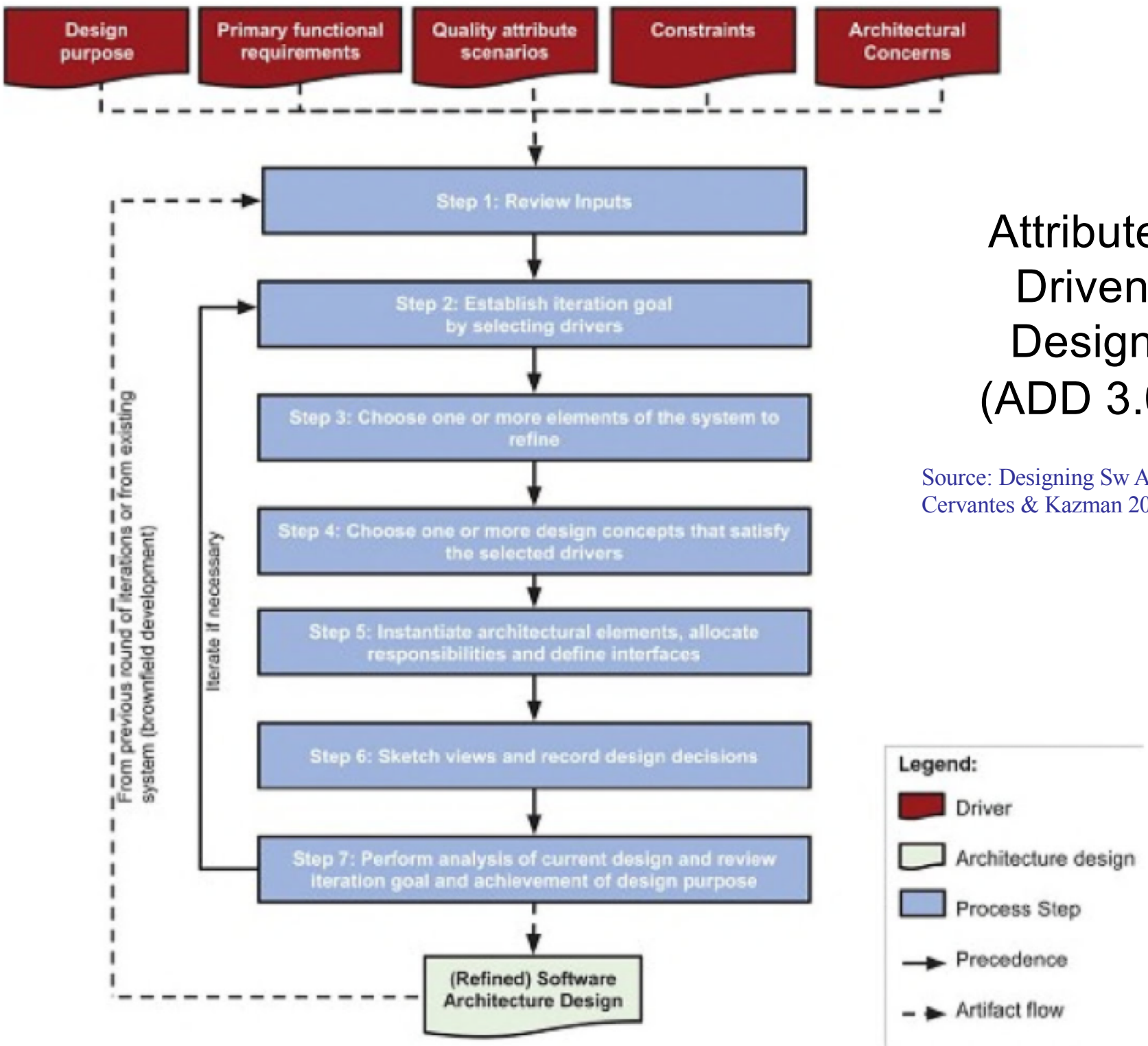
The role of architecture in sw lifecycle



Architecting enables requirements reuse

- Software reuse is the process of reusing existing artifacts (code, documents) in building a new system
- Typical reusable artifacts: libraries, standard components, user interfaces (eg. browsers), document structures (eg. layouts)
- Also requirements can be reused, if put in the context of a *domain* and a *reference architecture*

Attribute-Driven Design (ADD 3.0)



Source: Designing Sw Architectures, Cervantes & Kazman 2016

DevOps

- DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while insuring high quality
- The DevOps main practices are:
  - Engaging operations as a customer and partner, “a first-class stakeholder”, in development. Understanding and satisfying requirements for deployment, logging, monitoring and security in development of an application.
  - Engaging developers in incident handling. Developers taking responsibility for their code, making sure that it is working correctly, helping (often taking the role of first responders) to investigate and resolve production problems.
  - Ensuring that all changes to code and configuration are done using automated, traceable and repeatable mechanisms – a deployment pipeline.
  - Continuous Deployment of changes from check-in to production, to maximize the velocity of delivery, using these pipelines.
  - Infrastructure as Code. Operations provisioning and configuration through software, following the same kinds of quality control practices (versioning, reviews, testing) as application software.

Microservices and DevOps

- DevOps work is done by small agile cross-functional teams solving end-to-end problems independently, which means that they will build small, independent services
- microservices introduce many points of failure; resilience has to be designed and built into each service. Services cannot trust their clients or the other services that they call out to. You need to anticipate failures of other services, implement time-outs and retries, and fall back alternatives or safe default behaviors if another service is unavailable. You also need to design your service to minimize the impact of failure on other services, and to make it easier and faster to recover/restart.
- Microservices also increase the cost and complexity of end-to-end system testing; run-time performance and latency degrade due to the overhead of remote calls.
- monitoring and troubleshooting in production can be much more complicated, since a single action often involves many microservices working together (an example at LinkedIn, where a single user request may chain to as many as 70 services).

Conclusions (from R. Kazman)

- If you are building a large, complex system with relatively stable and well understood requirements and/or distributed development, doing a large amount of architecture work up-front will likely pay off.
- On larger projects with unstable requirements, start by quickly designing a candidate architecture even if it leaves out many details. Be prepared to change and elaborate this architecture as circumstances dictate, as you perform your spikes and experiments, and as functional and quality attribute requirements emerge and solidify.
- On smaller projects with uncertain requirements, at least try to get agreement on the major patterns to be employed. Don't spend too much time on architecture design, documentation, or analysis up front.

Self-test questions

- What is the role of architecture in a traditional, waterfall process ?
- Describe and compare the use of architecture in Waterfall, Iterative and Agile processes
- What is the relationship between functional requirements and architecture?
- What is the relationship between non-functional requirements and architecture?

References

- Cervantes and Kazman, *Designing software architectures*, AW 2016
- Coplien & Bjørnvig, *Lean Architecture*, Wiley, 2010
- Babar & Brown, *Agile software architecture*, MK 2013
- Schmidt, *Architecture-driven Software Development*, Elsevier, 2013
- Bass, Weber, and Zhu, *DevOps: A Software Architect's Perspective*, SEI, 2015

Useful sites

- www.infoq.com/articles/agile-software-architecture-sketches-NoUML
- www.sei.cmu.edu/architecture/tools/define/add.cfm
- www.agilearchitect.org
- www.holistic-software.com/agile-architecture

Questions?