# Comparing Development Processes

Peter C Rigby

# Development Processes

- Code and fix
- Waterfall
- Spiral Model
- Agile
- RUP
- The Eclipse Way
- OSS and Linux

# Code and Fix

- Code for a while
- Then fix bugs...

- Adhoc development important in new field
  - Determine new methodologies and processes

- Why use this model?
- Why not?

# Code and fix

- Little or no time for planning
- Other stages are combined into the fix stage
  - e.g., analysis, design, testing
- Alternative solutions are not seriously considered
- Code becomes unmaintainable quickly with many ripple effects
- But, time pressures mean that quick fixes are still done in software industry

# Cost of fixing errors

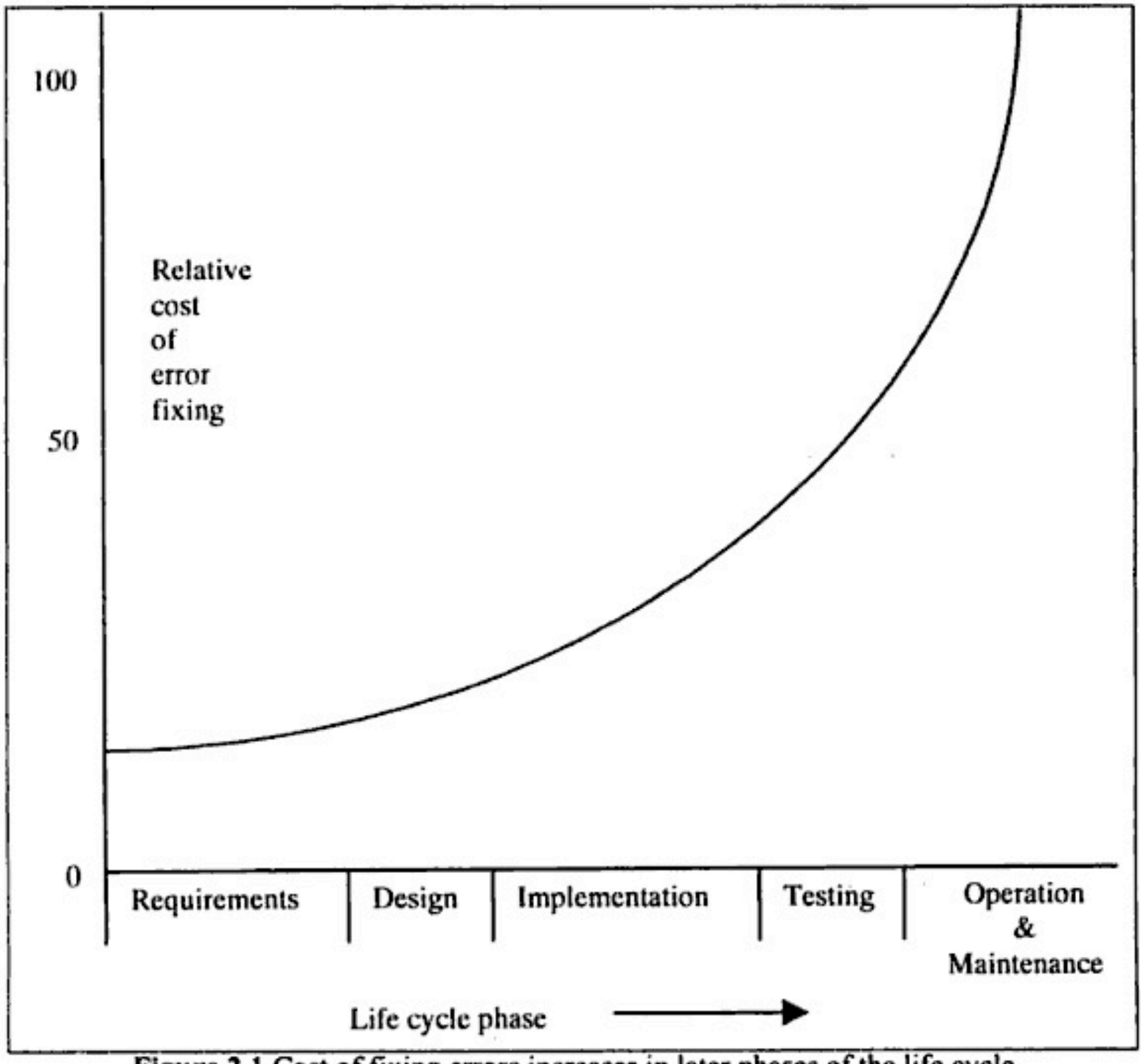
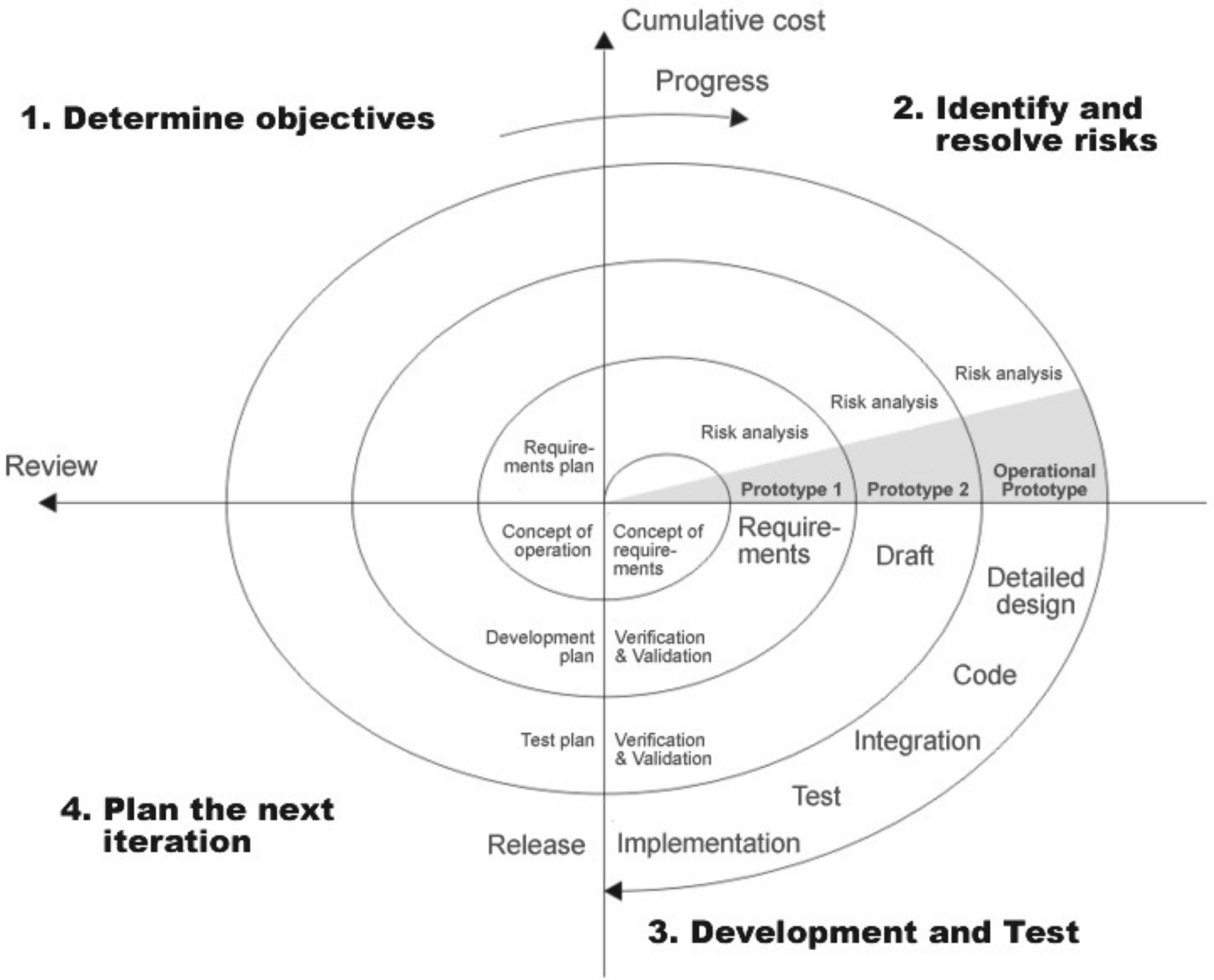



**Figure 2.1** Cost of fixing errors increases in later phases of the life cycle

# Waterfall

- Typical problem solving paradigm:
  - Decide what to do (requirements)
  - Decide how to do it (design)
  - Do it (implementation)
  - Test it (verification)
  - Use it (maintenance)
- Sequential flow – but later refinements did include feedback loops to earlier phases
  - Is any stage ever truly completed?
- Maintenance, leads to underestimation of evolution effort
  - Defects result from evolution, not just from, for example, misspecification

# Spiral model

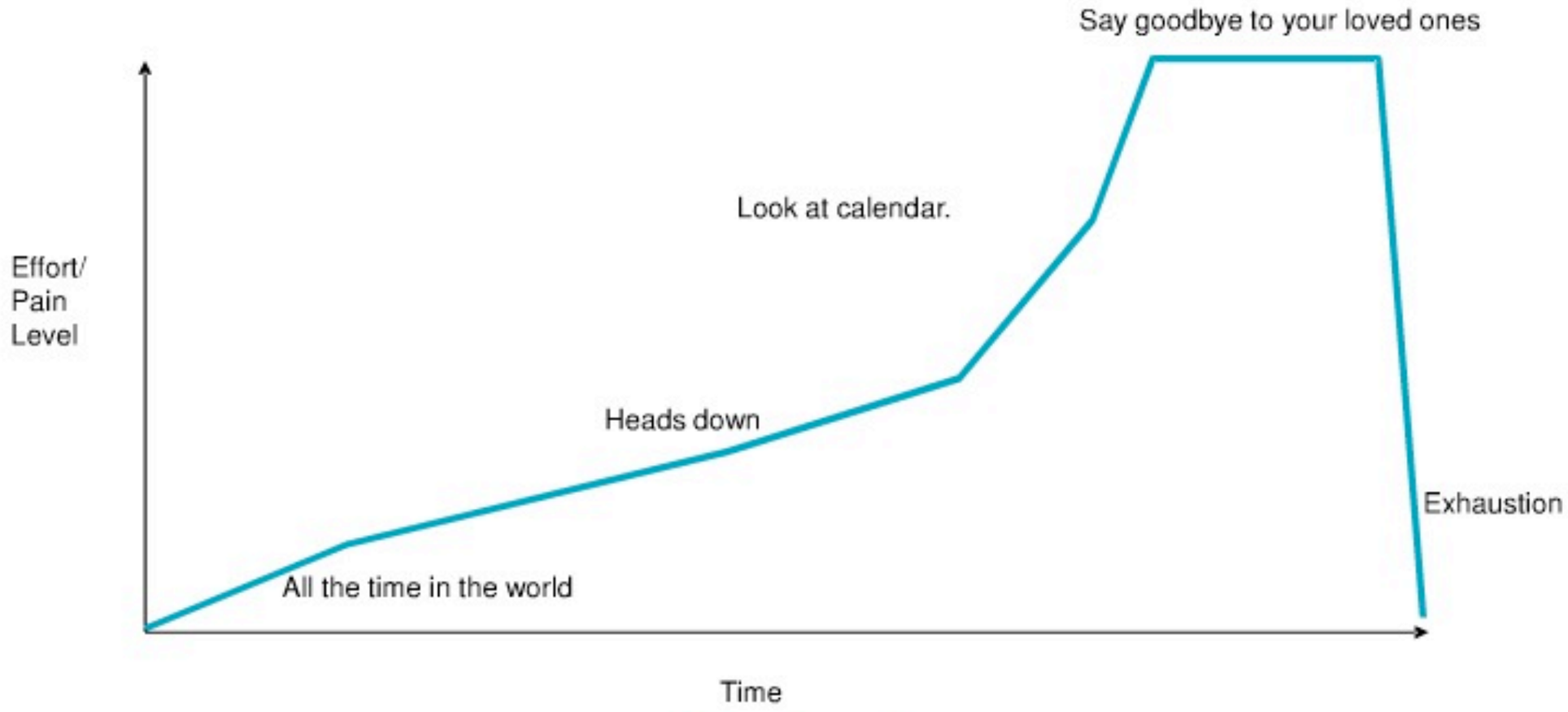


# Spiral model

- Phases in the model are defined cyclically
- 4 stage representation through which development spirals, at each level:
  - **Objectives**, **constraints** and **alternatives** are identified
  - Alternatives are evaluated, **risks** are identified and resolved
  - Next level of the product is **developed** and **verified**
  - **Next phase** is planned

# Spiral model

- Added prototypes, risk assessment, and iteration to waterfall model, but
- Iterations are between 6 months to 2 years long!
- That's a very long time before having a prototype that you can show your users
  - May be appropriate for well defined, high risk projects
- Evolution (e.g., changing user requirements and environmental factors) doesn't proceed that slowly




# Long VS short Iterations

# Paving the cow paths?

- The system simply automates the previous procedure in an exact manner
  - No consideration of efficiencies and possible usages or evolutions
- The process simply imitates previous engineering and ad hoc processes
  - But software is more malleable and intangible than traditional engineering disciplines, so it evolves more rapidly

Software models are part of reality and reality changes…

Are there software processes that take evolution into account from the beginning?

"Embrace Change", Kent Beck, XP

Try to find good existing patterns that work, … hunt for practices that work, try them out, if they work, keep them, if they don't, drop them

Paraphrase of Erich Gamma, Eclipse

"Release early. Release often. And listen to your customers.", Eric Raymond, Linux

# Agile Manifesto

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

# Ideas behind agile

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Even late changes in requirements are welcomed
- Close, daily cooperation between business people and developers

# Ideas behind agile

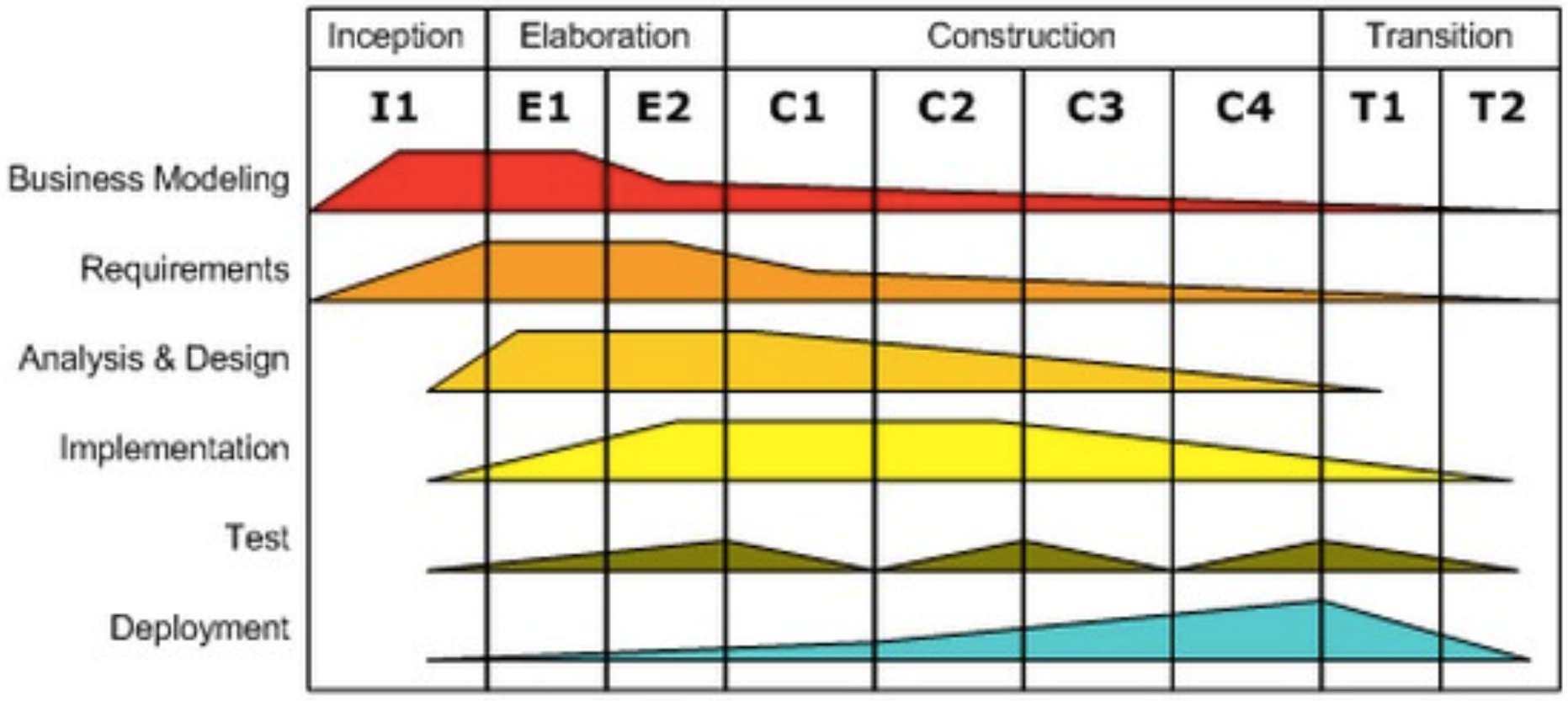
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstance

# Agile Methods

- Test driven development (TDD)
  - Write a failing test
  - Write or modify code until the test passes
  - Regression testing reduces the risk of ripple effects
- Behavior Driven Development (BDD)
  - Create user stories on flash card
  - Encourages collaboration between developers, QA and non-technical or business participants

# Agile Methods

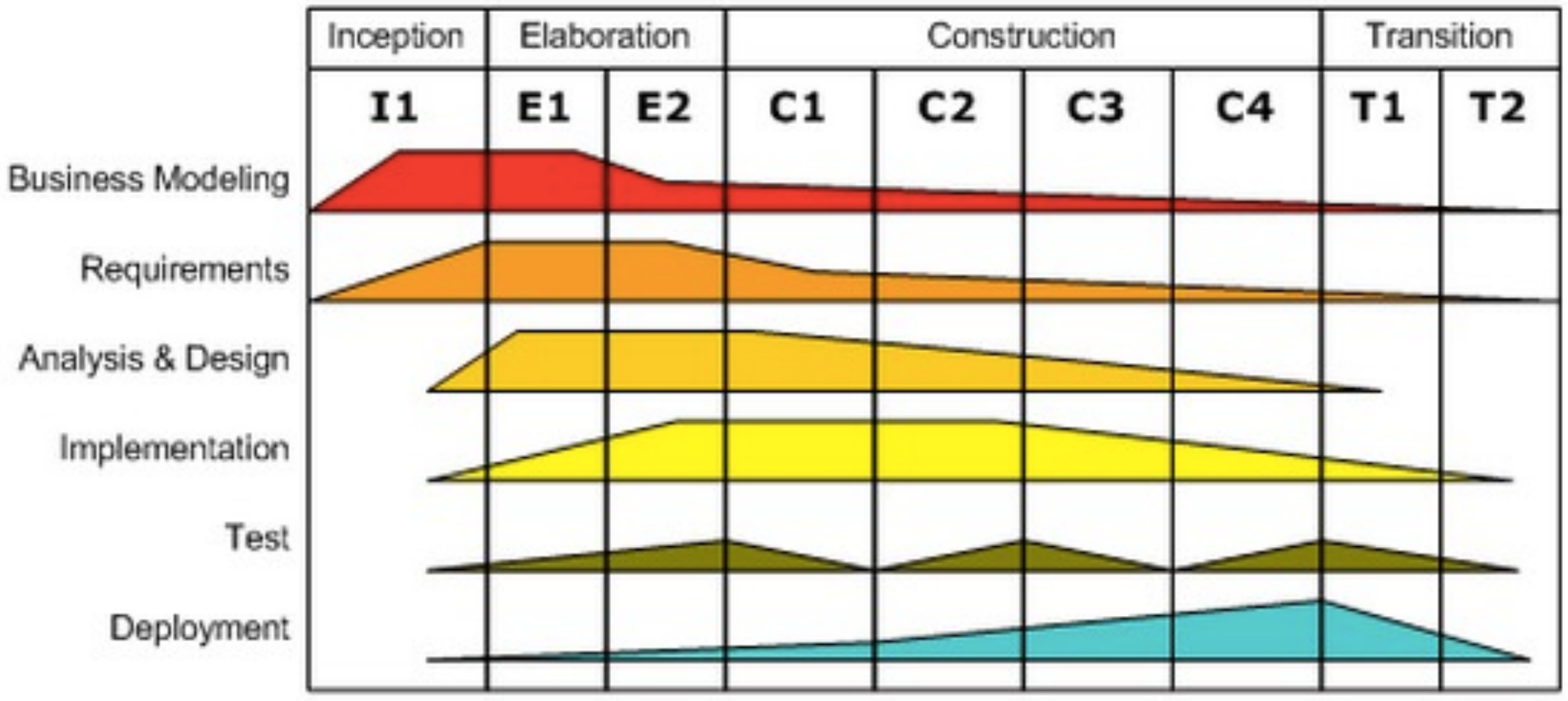
- Continuous integration
  - Always have a releasable system
- Pair programming
  - Pairs solve problems more quickly and are continuously reviewing each other
- Consensus based decision making
  - Empowerment and planning
- Continuous feedback from users
  - On-site business partner, Bugzilla, usergroups




# Unified Process

# Inception Phase

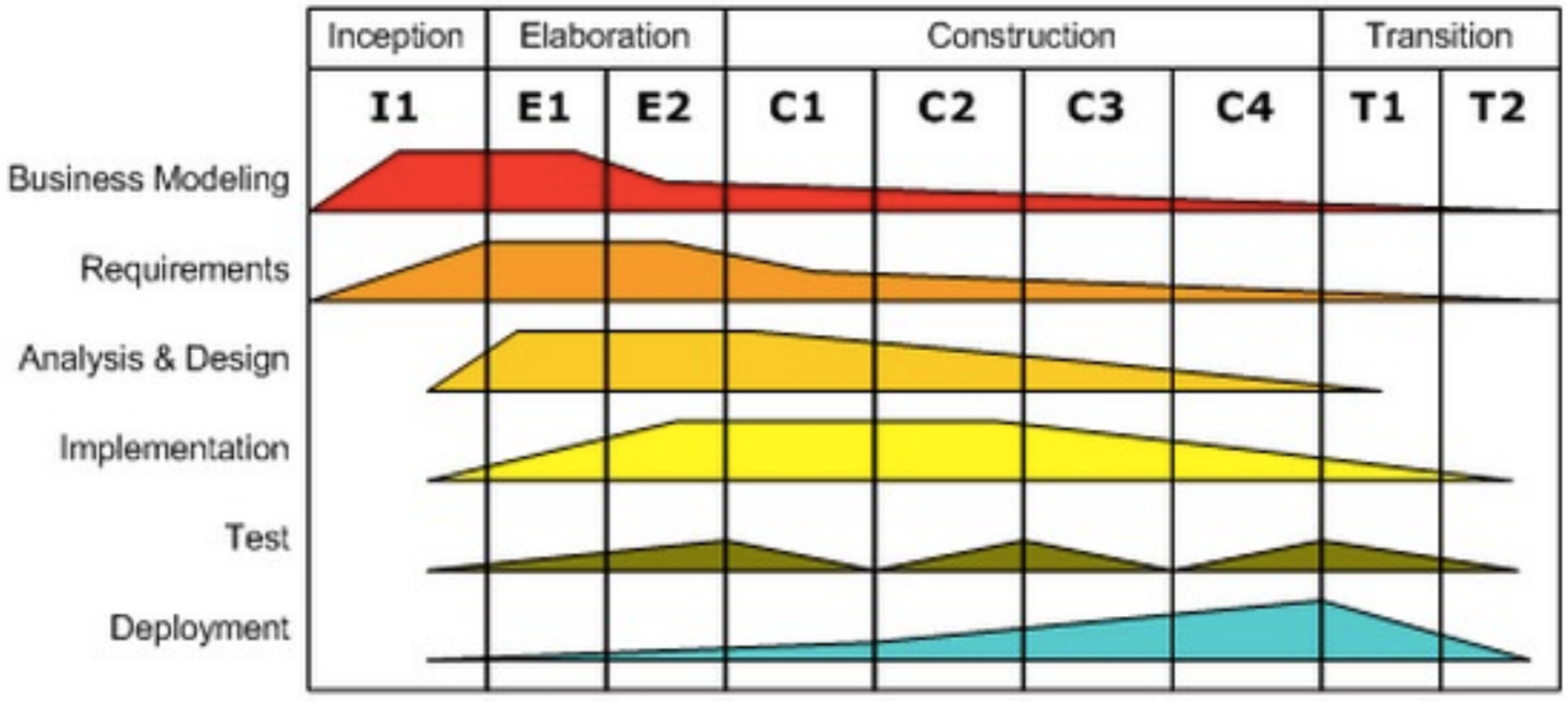
- Scope the project
- Business case for the project
- Key use cases (key requirements)
- Candidate architectures
- Risks, plans, and cost estimates
- Lifecycle Objective Milestone




# Unified Process

# Elaboration

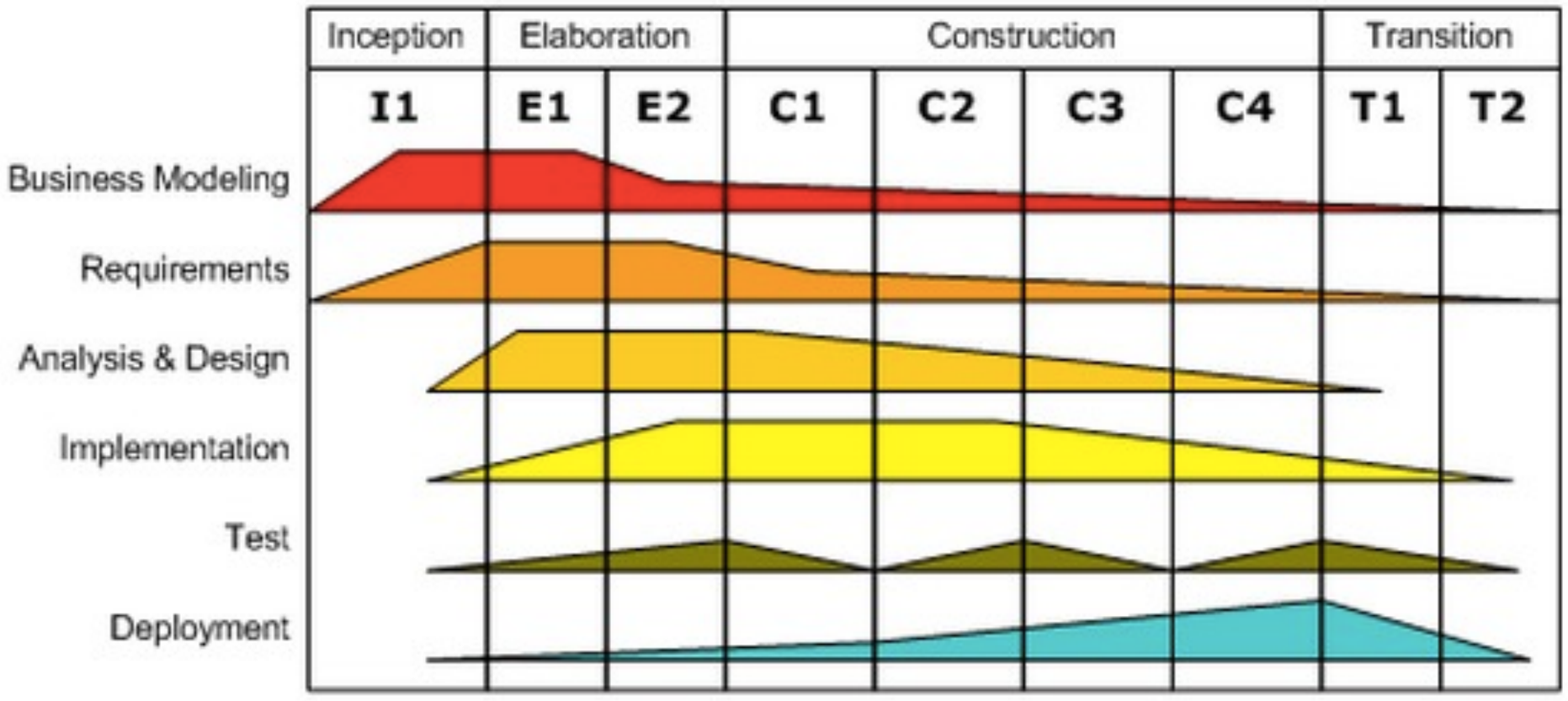
- Risk factors and architecture
- 80% of the use cases are finished
- Development plan
- Prototypes that mitigate identified risks
- Lifecycle Architecture Milestone




# Unified Process

# Construction Phase

- Coding, the longest phase of the project
- Fully elaborated use case starts iteration
  - May create multiple UML diagrams
- Divide into multiple components
- Release the system after each iteration
- Operational Capability Milestone




# Unified Process

# Transition Phase

- From development to production
- Training end users, maintainers, beta testers
- Checked against quality objectives
- Product Release Milestone

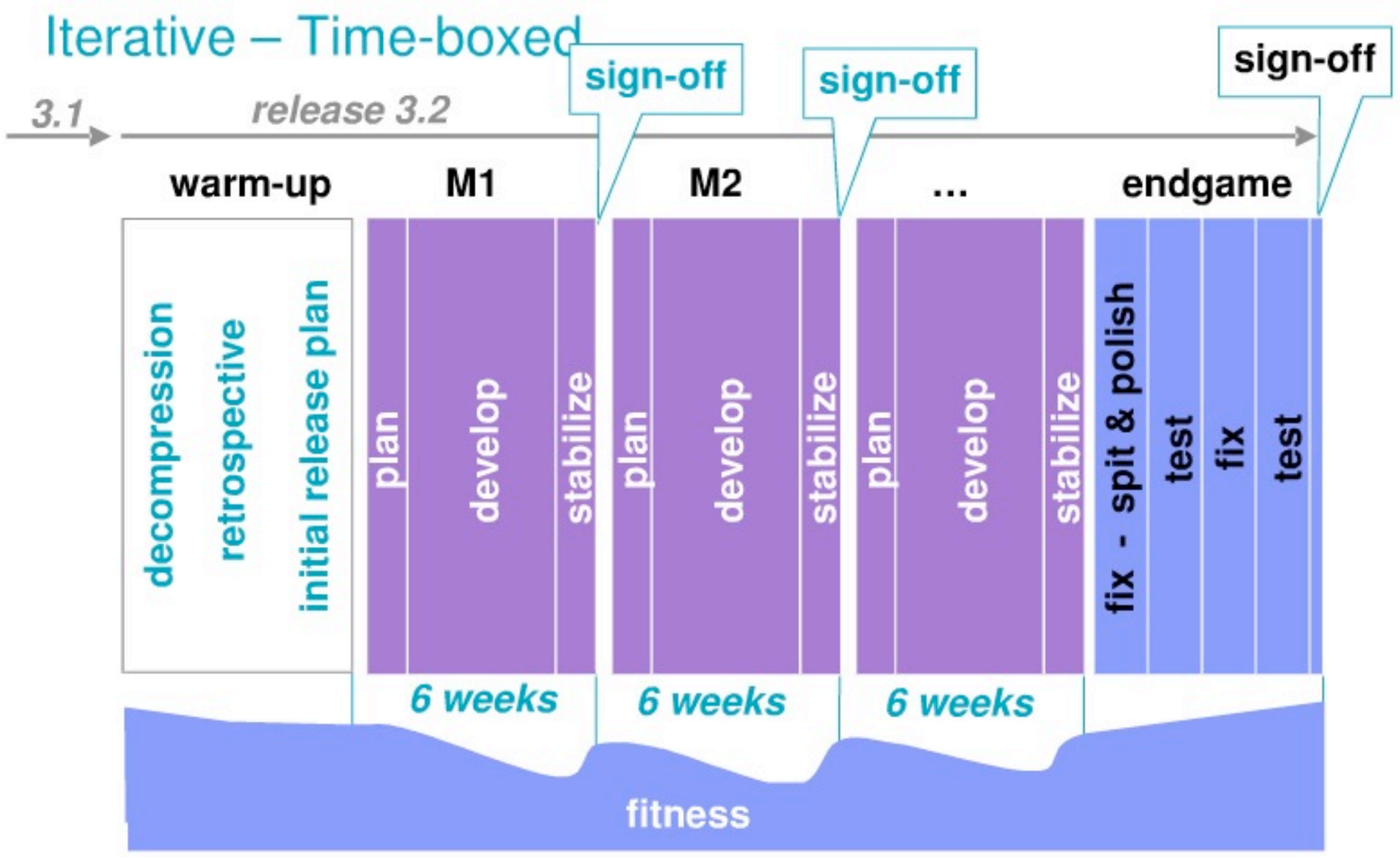
# RUP Six Best Practices

- Develop iteratively
- Manage user requirements
- Use components
- Model Visually
- Verify Quality
- Control changes

# The Eclipse Process

- Erich Gamma
- http://www.infoq.com/presentations/Eclipse-Lessons-Erich-Gamma
- Process starts at 27th minute
- Recommend that you watch the whole thing

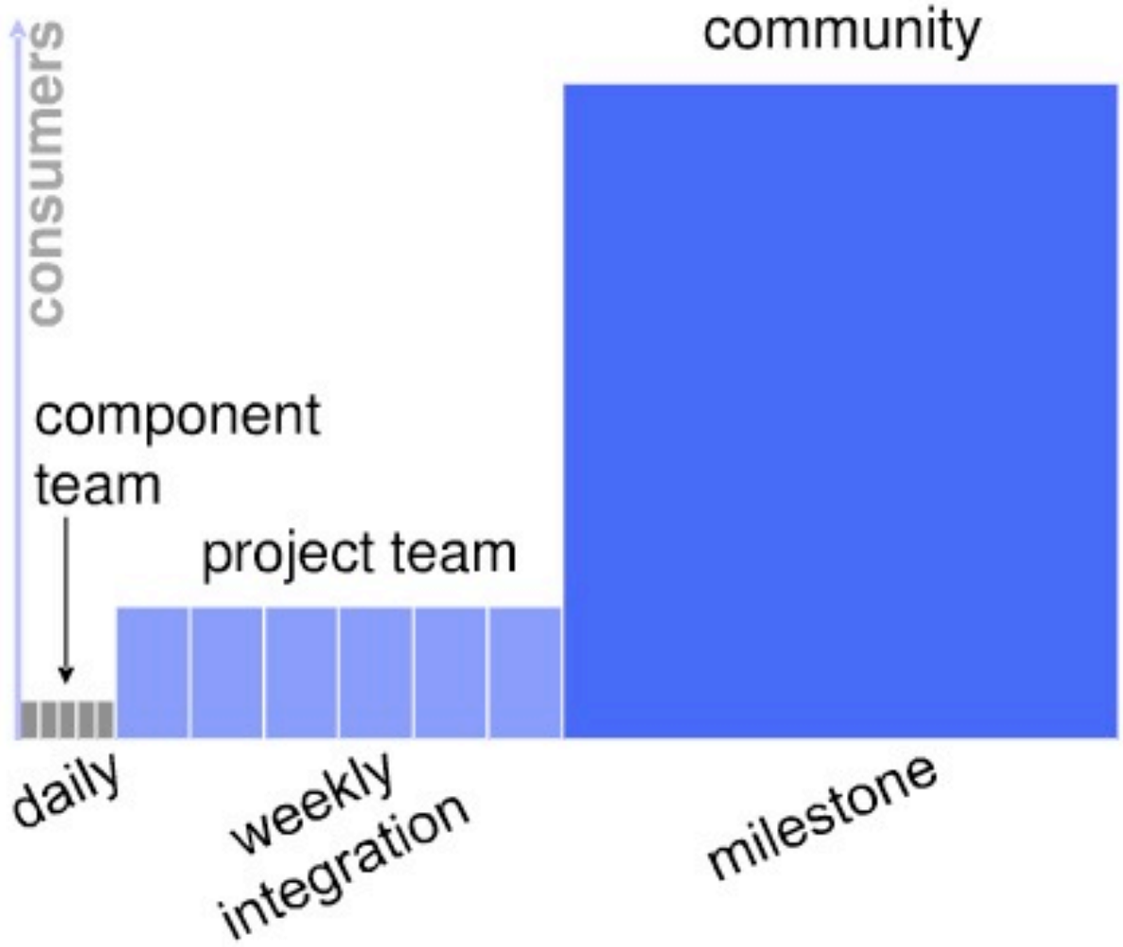
# Eclipse Model



# Building

## Builds

- continuously **consumable**
- continuously **interesting**
- continuous **listening**
  - users have influence
  - encourages feedback




- we continuously ***consume our own output***

# Eclipse model

- Very short intervals 4 – 6 weeks
  - Add value incrementally
- Integration, daily, weekly, …
  - Integrate more regularly with your team
- Endgame
  - Code freeze, stabilization process
- Publish milestones, always have a releasable system
  - You learn by shipping, so ship more often
- Accountability
  - Demos what you've done

# Cathedral vs Bazar

- Unix gospel
  - small tools,
  - rapid prototyping,
  - evolutionary programming for years
- But Raymond still believed in cathedral for large systems like operating systems

# Customers/Users

- 1. Every good work of software starts by scratching a developer's personal itch.
- 7. Release early. Release often. And listen to your customers.
- 6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.

# Iterate

- 2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
- 3. “Plan to throw one away; you will, anyhow.”, Brooks

# Quality Assurance

- "Given enough eyeballs, all bugs are shallow.", "Linus's Law".
- 8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.
- "Somebody finds the problem, and somebody else understands it. And I'll go on record as saying that finding it is the bigger challenge.", Tovalds

# Distribution

- 19: Provided the development coordinator has ... the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

# Is OSS Agile?

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

# What is the main difference?

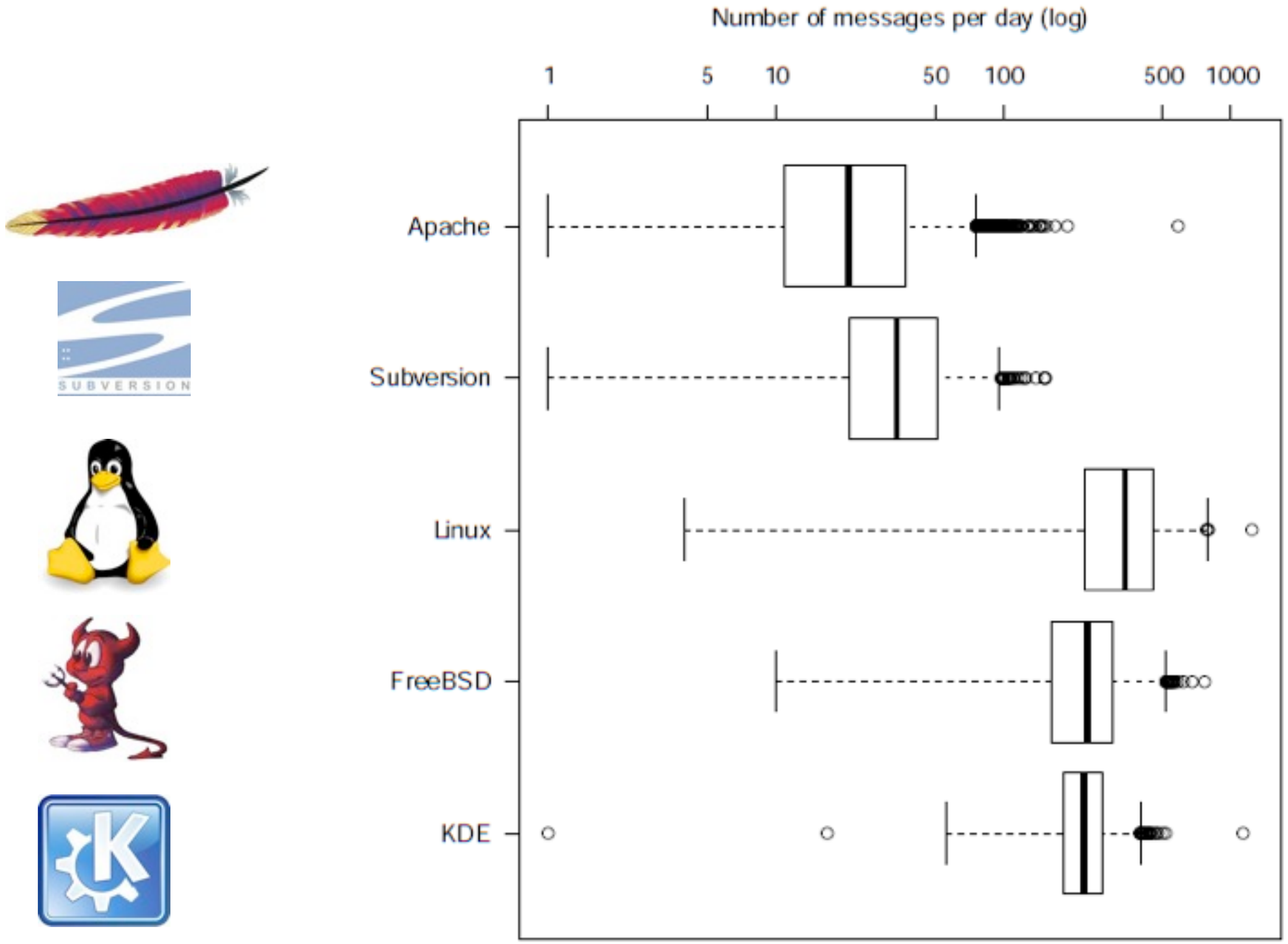
# Size and Location

- Agile is done by small, co-located teams of developers
- OSS scales to large, distributed teams of developers

# Osmotic Communication

- "information flows into the background hearing of members of the team, so that they pick up relevant information as though by osmosis", Cockburn

# Mailing list traffic



# OSS Techniques

- Minimal overhead processes
  - Keep process that work for your project
  - Get developer input
- Iterative and very small chunks of work
- Pay attention to your users and release early and often
- Meritocracy – status based on work
- Share all development effort with the community
  - Allows for distributed development in large groups

# Variance in Software Products

- Most of the variance in software engineering can be explained, not by the process, but by the expertise of the individuals
  - Don't slow down your most effective people with overly formal processes
- For example:
  - 10:1 difference between average and best programmers
  - The software inspection process does not predict the number of defects found, but the expertise of the inspectors does

# Summary

- Agile methods account for evolution from the beginning
- Iterate!
  - Don’t wait until the end to find out you’ve made a mistake at the beginning
  - Create small, independent, complete tasks that add value
- Don't get bogged down by process, use what works
  - Plan: Cowboy coding doesn't work!
- Involve your users and release products as early as