Intro to Software Process

How do you develop software now?

Stakeholders vs. Developer

Stakeholders:

- How much will it cost?
- How long will it take?

Developer:

- Develop on budget
- Develop on time
- Develop the ***"right"*** software

The Straightforward Approach

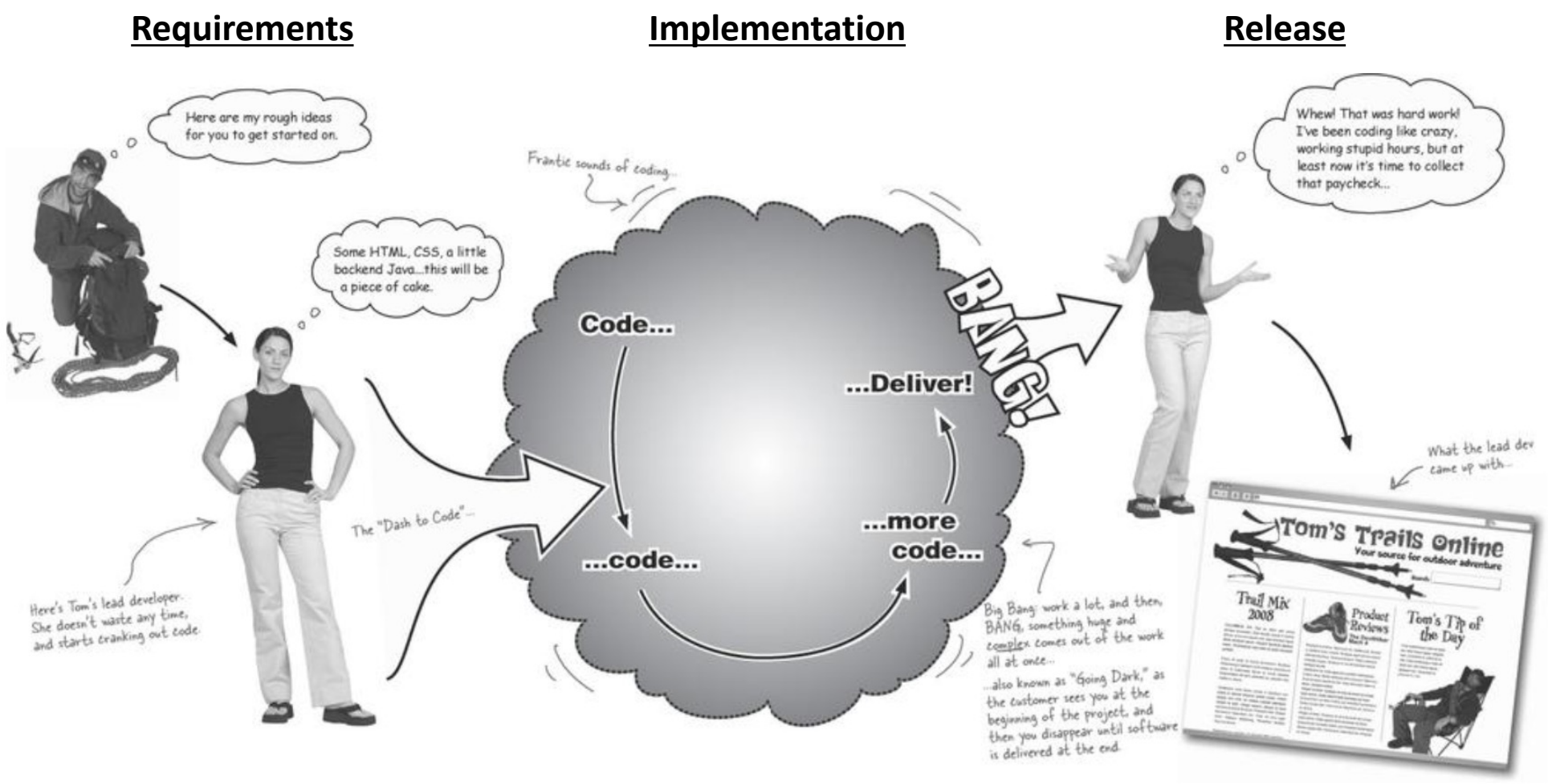


Image: Head First Software Development 2nd Edition

Not quite…

Standard Software Development Phases

- Requirements
- Analysis
- Design
- Implementation
- Testing & Validation
- Release
- Maintenance

Software Development Artifacts

- **Requirements** → *requirements specification*
  - E.g., user stories, use cases, all the way to 500-page requirements documents
- **Analysis** → *task identification*, *viability testing*
- **Design** → *technical specification*
  - E.g., Quick class diagram (UML), sequence diagrams, extensive system design documentation, plus more…
- **Implementation** → *code*
- **Testing** → *test report, e.g., set of test cases, testing/acceptance report*
- **Release** → *executables*, e.g., packages, installers
- **Maintenance** → *patches, hotfixes, updates (distribution methods vary)*

Waterfall

- *Heavyweight*, *single-pass*, heavy *planning*-based approach
- Each phase completed and verified before the next phase begins
- “Big Bang” Approach as described by the Head First book
- Requires **extreme amounts of planning** to be successful

Why Waterfall?

- A response to the **software crisis** (~1960s):
  - Projects running over-budget
  - Projects running over-time
  - Software was very inefficient
  - Software was of low quality
  - Software often did not meet requirements
  - Projects were unmanageable and code difficult to maintain
  - Software was never delivered
- Makes sense at first
- The predominant process for other engineering disciplines

Waterfall Weaknesses

- Assumes no overlap in phases
- Heavily disrupted by volatile requirements
  - Requirements are almost always **not** fully known in advance, and often added during the other phases
  - Microsoft found that 30% of requirements originated after requirements gathering
  - Any design based on requirements documents must be considered temporary
- Not proved to be successful or produced low-quality software
  - Success is a matter of “luck”

Workarounds: Design for Change

- Since requirements are volatile, just design to allow for future changes
- Commonsense, but does require predicting the future, which we cannot always do
- Very useful for low-level design: named constants, good method abstractions, good class abstractions
- Increases flexibility, but that often increases complexity
- Increases cost of the initial software
- Fails to account for changes external to the software that affect the process

Workarounds: Prototyping

- Difficult to determine requirements without some sort of running system, so build a prototype to get requirements, e.g., “Build one to throw away”
- Develop the first version cheaply: Very high-level languages, fewer features (particularly non-functional requirements of speed, robustness, quality)
- Still necessary to gather all requirements during the prototype
- Problem of perception that prototype as a completed system

The Iterative Software Development Paradigm

Agile Software Development

- Lightweight, multiple pass, better balance between planning and implementation
- Strongly iterative and evolutionary
- “World is fundamentally chaotic”, “Change is inevitable”, “Deliver value to the customer”

Tenants of Agile Process

- Individuals and interactions are more important than processes and tools
- Working software is more important than comprehensive documentation
- Customer collaboration is more important than contract negotiation
- Responding to change is more important than following a plan

Waterfall vs Iterative Process

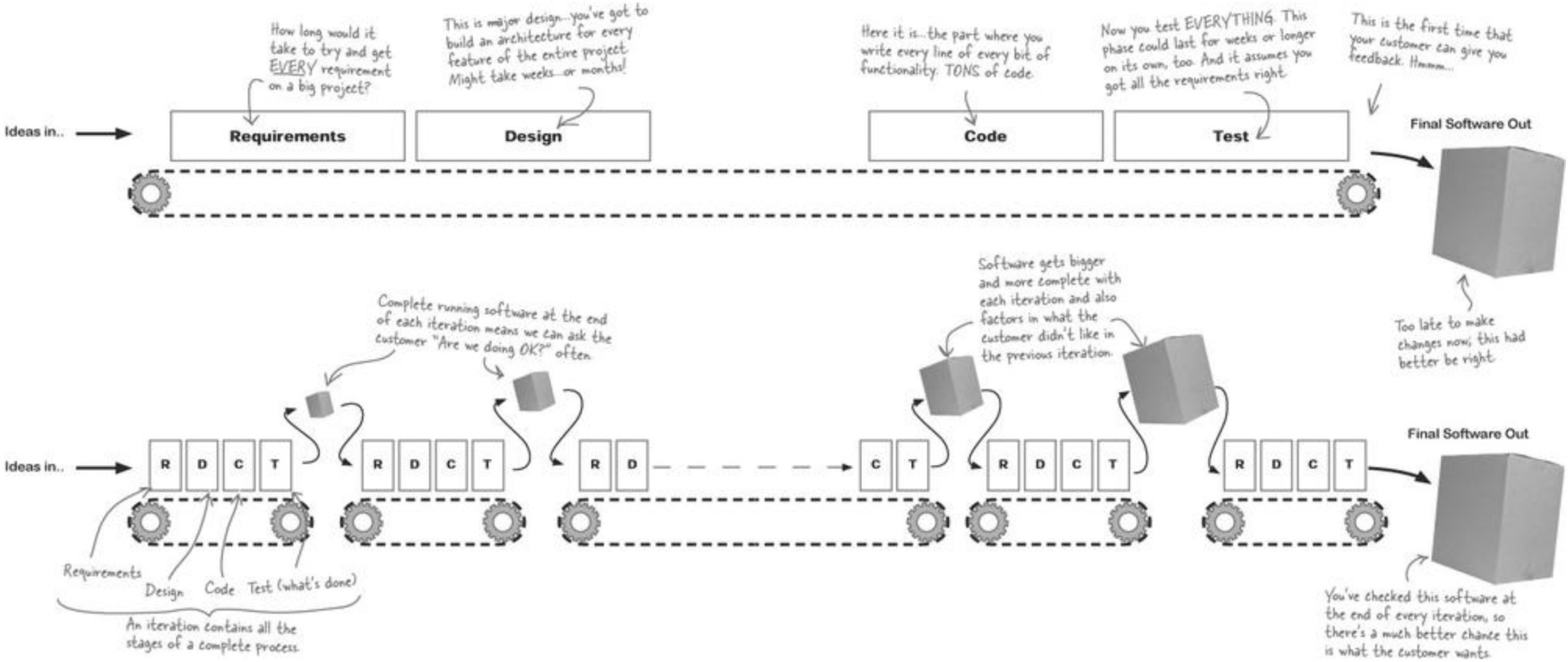
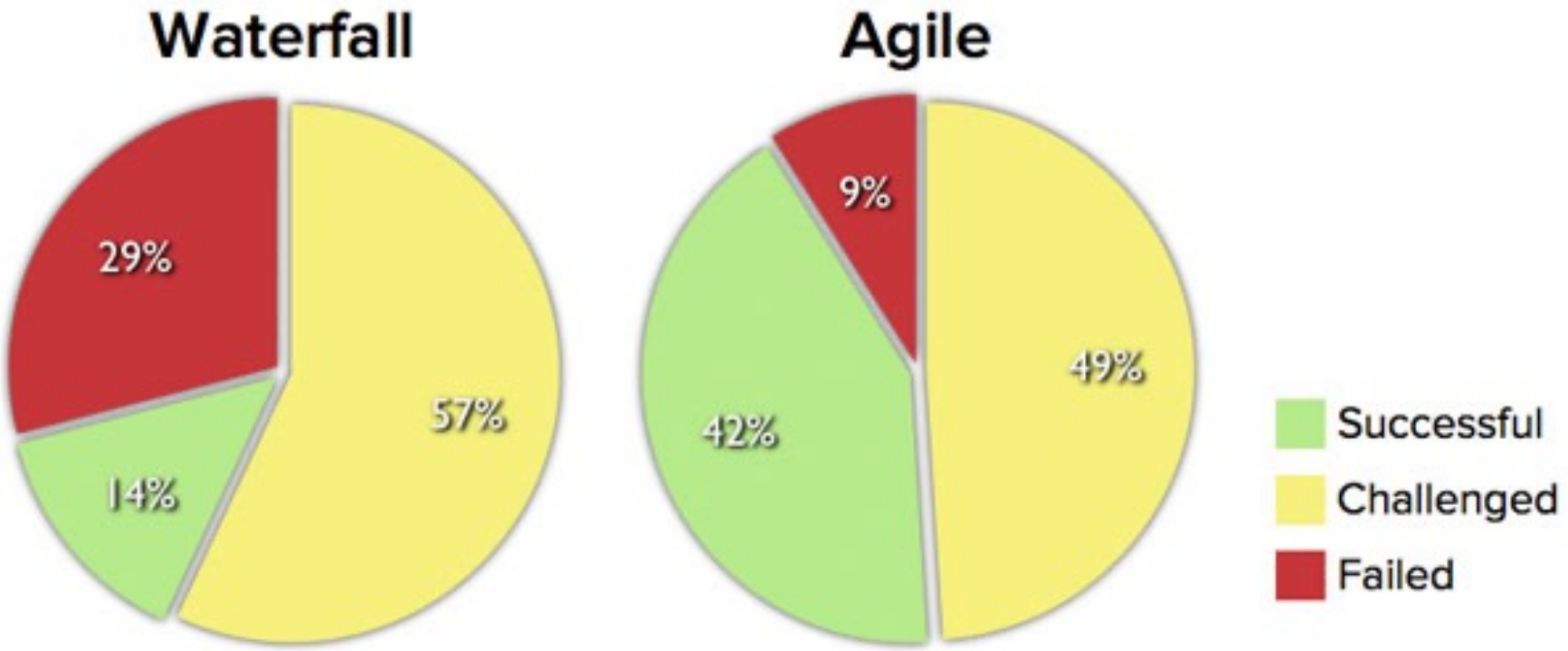


Image: Head First Software Development 2nd Edition

Don’t Take My Word For It…



Source: The CHAOS Manifesto, The Standish Group, 2012.

Requirement Changes with Iterations

- New requirement is given
- Working with stakeholders to prioritize new requirements with respect to the others
- Examine current iteration and adjust based on customer priority **AND** time remaining
  - If you have time, move things to a later iteration
  - If you don’t, stakeholders need to either allow for more time or make cuts