Agile Software Engineering

Mohammad Rezaei

January 2013

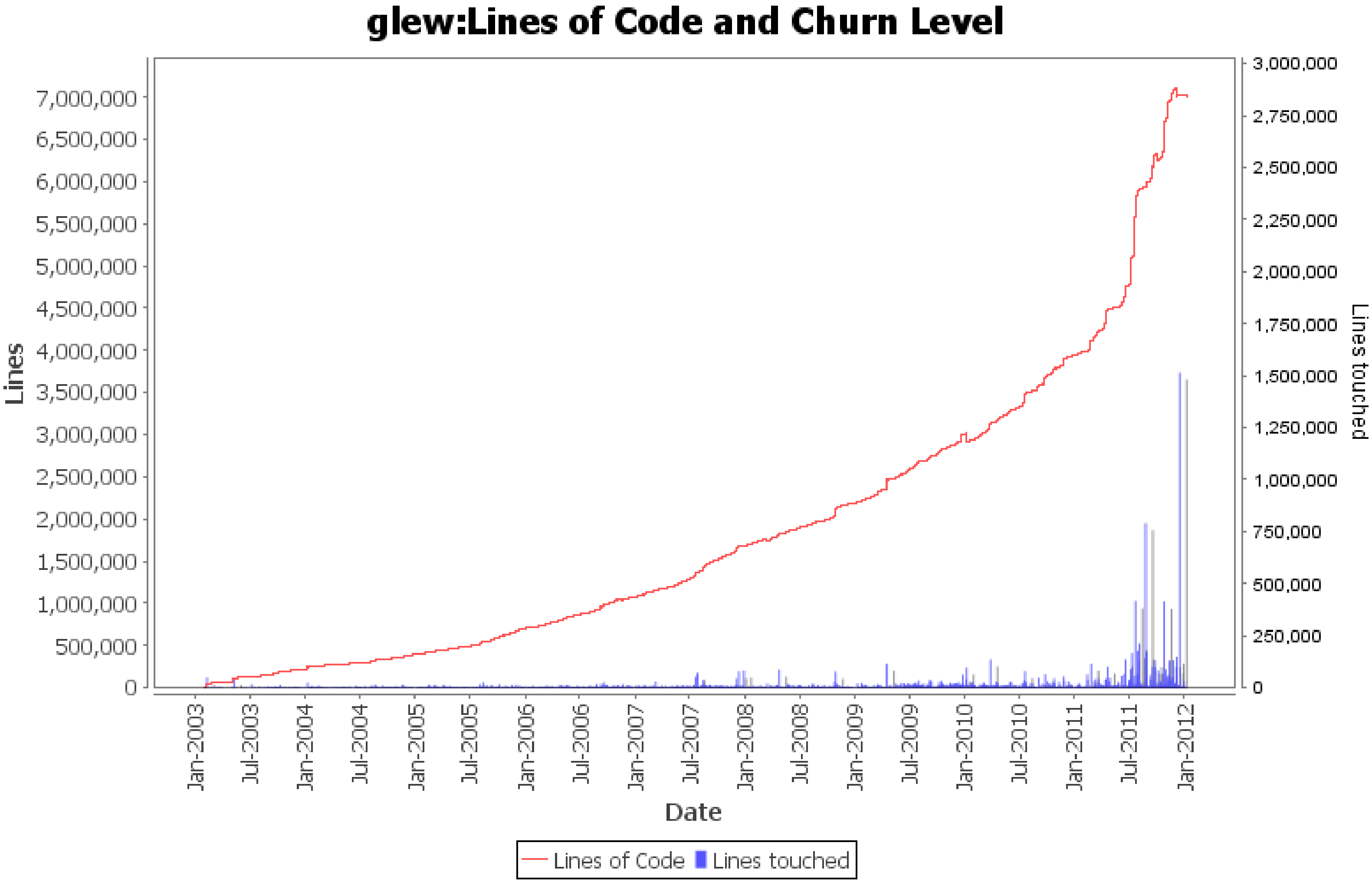
Overview

- Software engineering in the real world.
- What was there before Agile?
- What is Agile and how is it different?

Software Engineering in the Real World

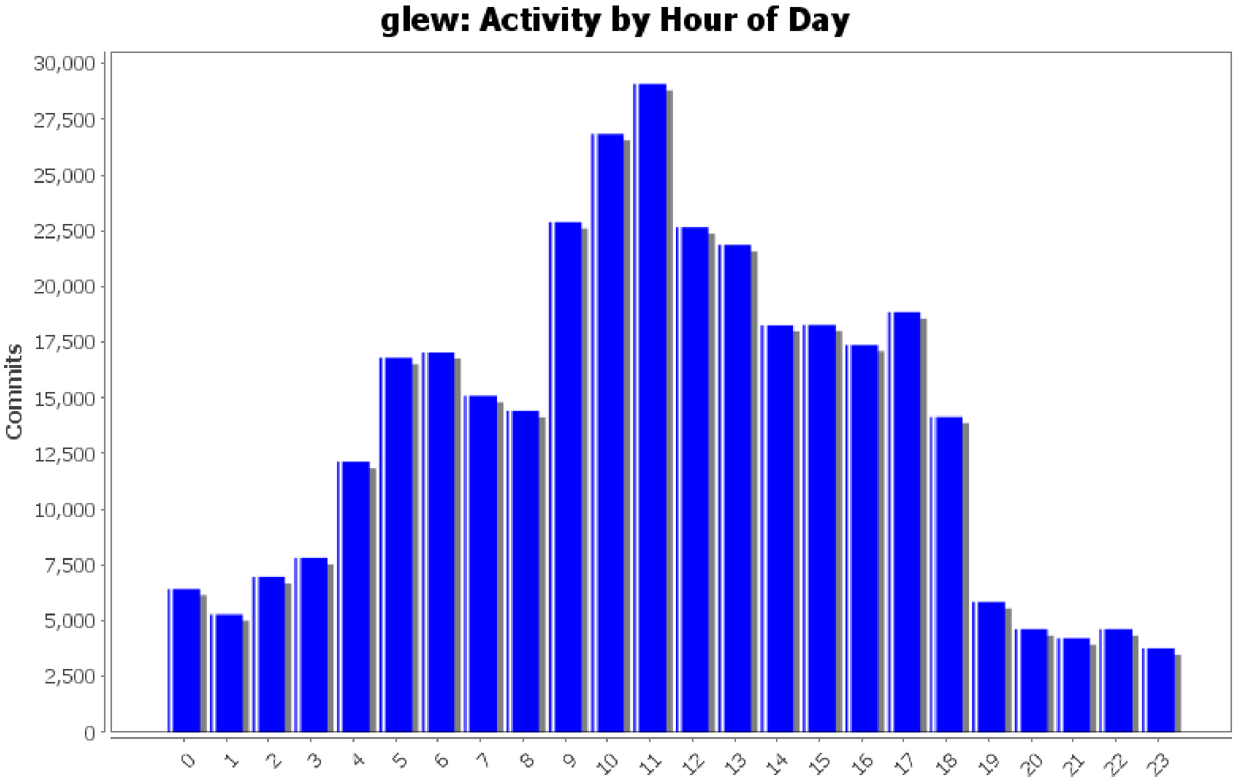
- IEEE (via Wikipedia): "Software Engineering (SE) is the application of a **systematic**, **disciplined**, **quantifiable** approach to the development, operation, and maintenance of software...“
- Coined in 1968.
- A short detour: an example of a real world system.

GLEW Project Size



GLEW Commit History

- 2.4 million lines of code.
- 340+ developers over past 9 years in 5 time zones.
- Only 60-70 active developers in a given month.



Real World Concerns

- Large existing code base.
- Large rate of change.
- Large team with significant turnover.
  - Nobody knows all the code
- Rapidly changing environment and requirements.
- Production code is not head code.
  - Release management.
  - Deployment into a live environment.
  - Support of the running system.
  - Auditability: reproduce old builds
- One constant: source control.

Back to Software Engineering…

- Wikipedia: "Software Engineering (SE) is the application of a **systematic**, **disciplined**, **quantifiable** approach to the development, operation, and maintenance of software...“
- Systematic:
  - Different people doing the same thing the same way.
  - Doing the same thing the same way at different times.
- Disciplined:
  - Apply the practices uniformly and consistently.
- Quantifiable:
  - Measure different aspects of the development process:
    - Technical aspects (e.g. test coverage).
    - Economic aspects.
      - Time and money spent.
      - Functionality delivered.

Software Engineering

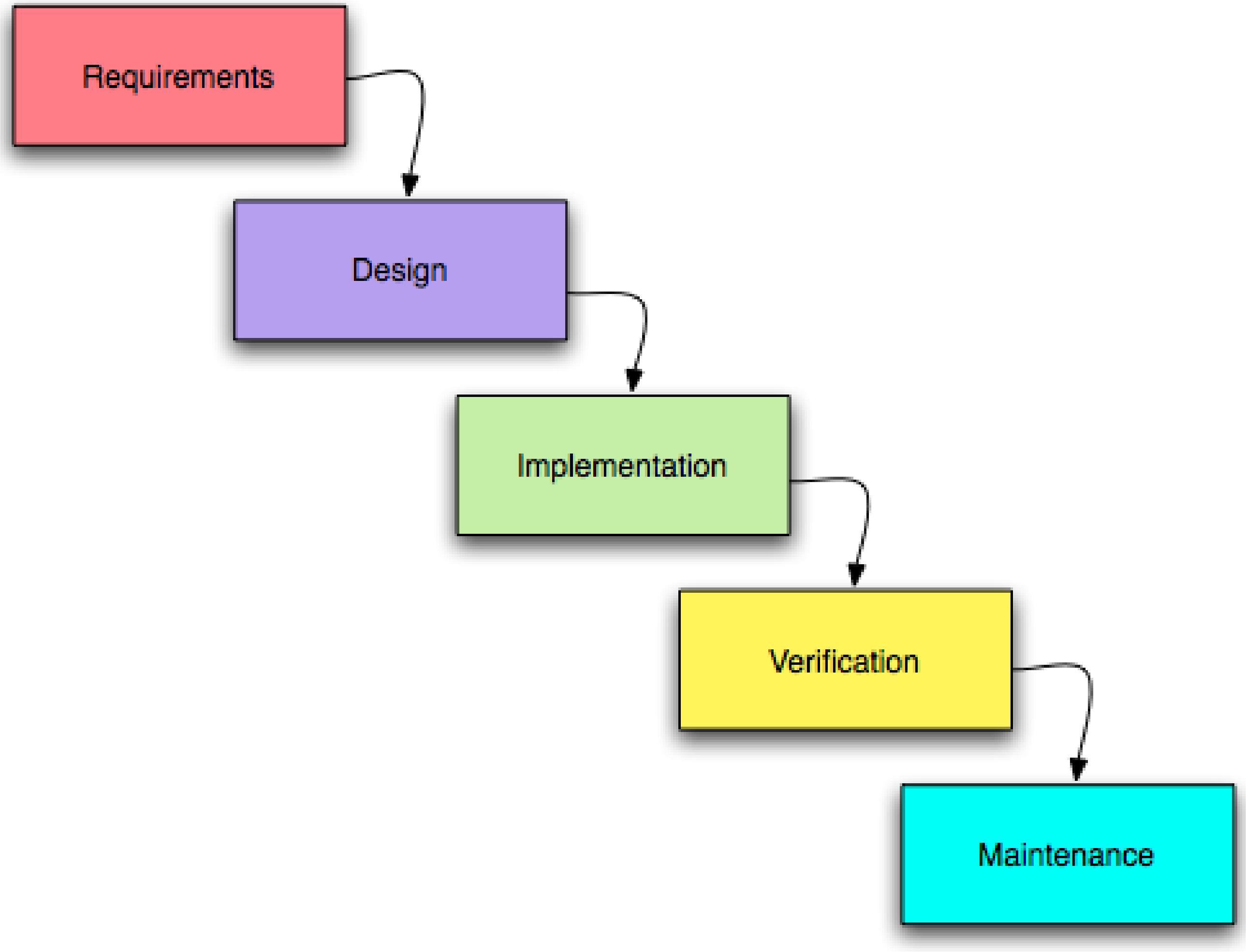
- Not really like most other engineering disciplines (e.g. civil, electrical, mechanical):
  - No real rules or formulae.
  - No legal standards for design and implementation.
  - No certifications.
  - Cost of fabrication is very different.
  - Closer to social engineering in many regards.
- Why do we need it?
  - In most cases it's about managing cost.
  - Three axes of cost
    - Quality
    - Scope
    - Time
  - Manage risk & uncertainty
    - Improve predictability
    - Optimize productivity
  - Different projects need different amounts and types of SE
    - Not going to discuss: sometimes lives are at stake (e.g. heart-lung machine, flight avionics)

Mental Challenge

- Rebuild GLEW from scratch!
- Plan the interaction of 340 developers over 10 years, producing 7 million lines of text.
- Take into account all the requirements by engaging the users.
- How would you go about doing this so it’s systematic, disciplined and measurable?

What was there before Agile?

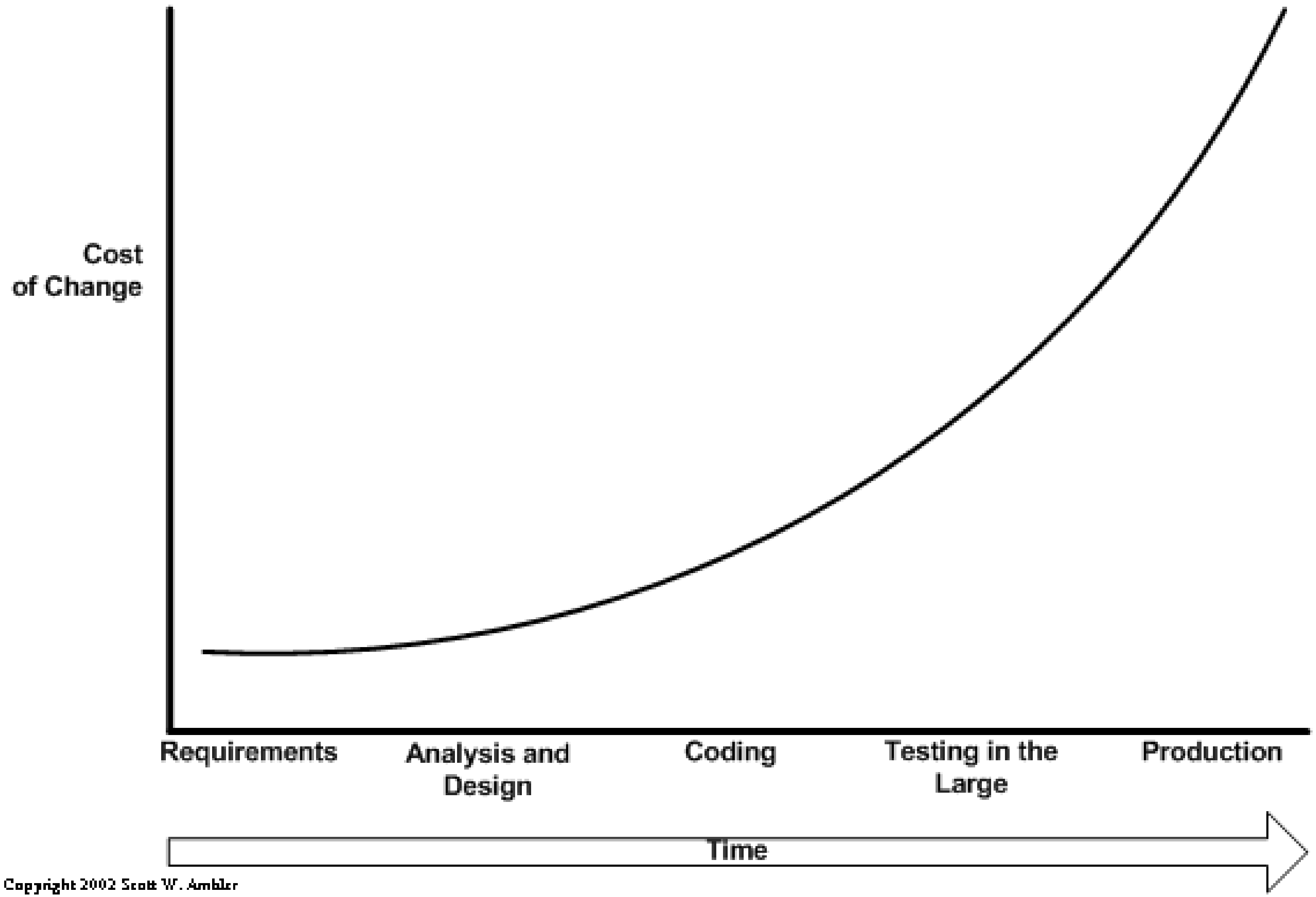
- Waterfall (image courteously of Wikipedia)



Problems with Waterfall

- A lot of work in design and implementation that was thrown out.
  - Because of changes to the project.
  - Because of simple miscommunication.
- Reams of documentation nobody ever read.
  - Documentation that was outdated very quickly.
- The feedback in waterfall is really one or at best two levels.
  - By the time we get to testing, the architecture is long done.

Cost Consequences of Waterfall



Agile Software Development

- Agile was a direct reaction to failures in software development that used long term, planned methodologies, such as Waterfall.
- Agile has a set of ***values*** and ***principles***.
- There are different implementations of Agile that incorporate these values and principles.
  - Extreme Programming
  - Scrum
  - Kanban
- It is useful to distinguish Agile practices that are technical vs. managerial in nature.

Agile Values

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

Agile Principles

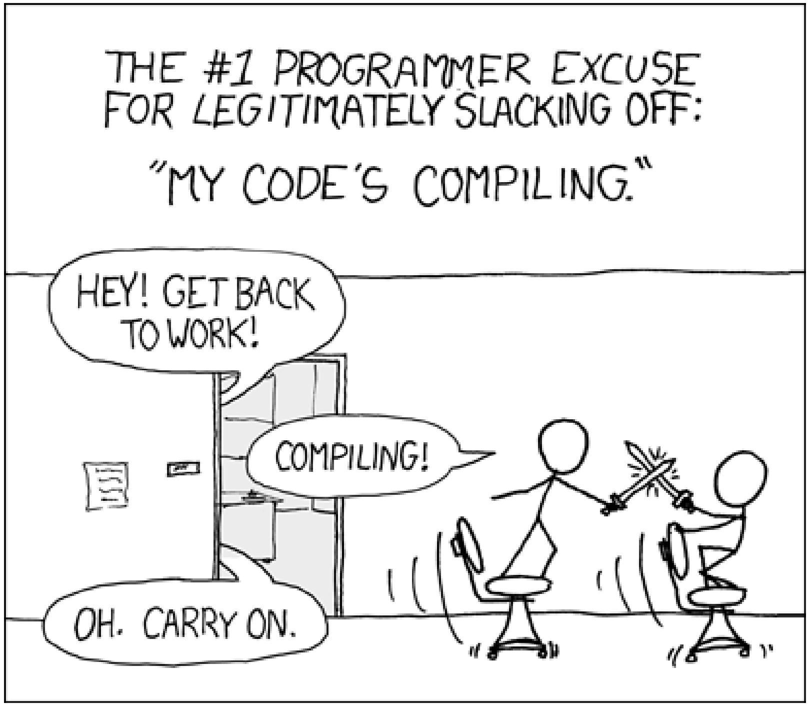
- Highest priority is customer satisfaction
- Welcome changing requirements
- Frequent delivery of software
- Business people & developers cooperating daily
- Build projects around motivated people
- Face-to-face conversation is best
- Progress measured by working software
- Sustainable development pace
- Continuous attention to technical excellence
- Simplicity
- Self-organizing teams
- Regular reflection & adaptation

Agile Fundamentals

- Flexibility with broad strokes
- Short feedback cycles
  - Continuous improvement
  - Human and machine provided
    - Automated tools that collect metrics
- YAGNI: you ain't gonna need it
- Sustainable pace
- Self-organizing, cross-functional teams.
  - Good mix of people, like a baseball team

Agile Technical Practices

- Automated Testing
  - Continuous build
  - Remote runs
  - Test Driven Development (TDD)
- Refactoring: reorganizing code without changing its function
  - Only viable with good automated tests
- Pair programming
- Automation
  - IDE: testing, refactoring
  - Continuous build
- Keep existing working practices
  - Consistent code style
  - Source control
  - Code reviews



Copyright XKCD

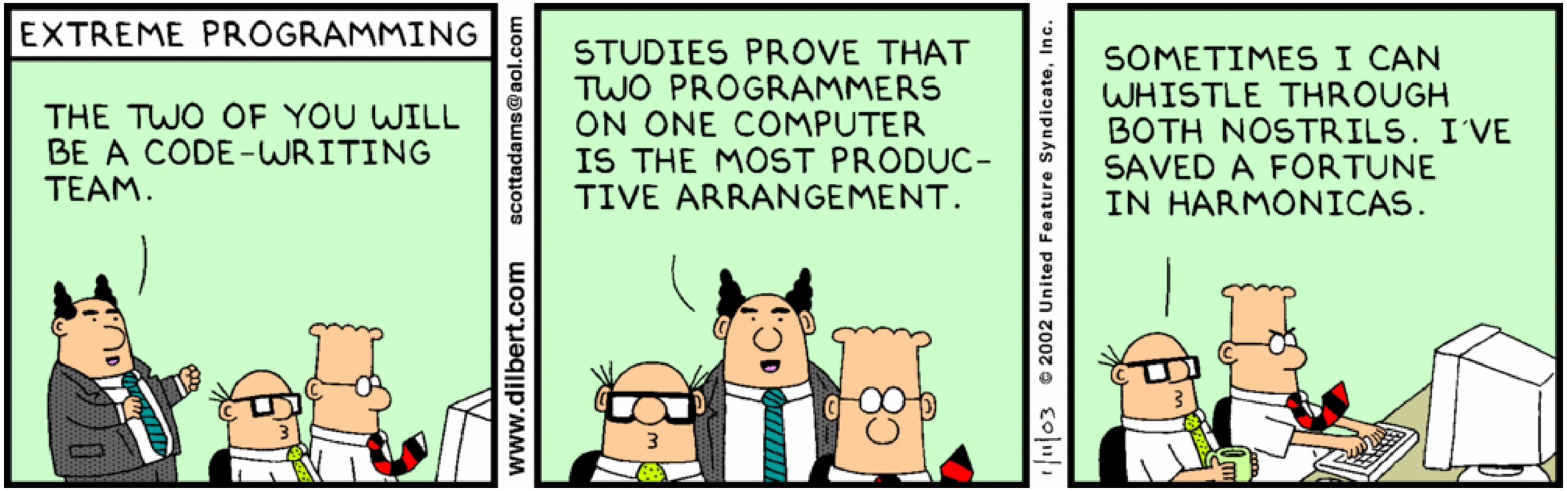
Automated Testing & Continuous Build

- Core principle: code that's checked into source control must be accompanied by automated tests.
- At face value, it sounds counter productive! After all, writing tests requires extra effort.
- The value proposition of automated tests are long term gains.
- The long term gains easily offset the short term loss.
- As a system grows, nobody can predict the effect of changing existing code.
- Automated tests lower the cost of change, because they remove the fear factor.
- In turn, that allows the developers to be more responsive to changing requirements.
- Continuous build automates the running of tests to provide further benefits:
  - Shortened QA because most bugs are caught early. This further leads to more frequent delivery.
  - Better collaborative environment because developers feel safe to update their code frequently as well as check in code frequently.
  - Shortened feedback cycle for developers: there is no need to wait for a QA team or QA period to find and fix issues with new or changed code.
- Automated tests have many additional benefits: increased quality, etc.

Agile Management Practices

- Iterations: small & incremental.
- Just in time estimates.
- User stories
- Retrospectives
- Stand up meetings
- Burn down charts
- Measured velocity

Agile: Customize the Practices to Your Environment


EXTREME PROGRAMMING
THE TWO OF YOU WILL BE A CODE-WRITING TEAM.
scottadams@aol.com
www.dilbert.com
STUDIES PROVE THAT TWO PROGRAMMERS ON ONE COMPUTER IS THE MOST PRODUC-TIVE ARRANGEMENT.
© 2002 United Feature Syndicate, Inc.
1/11/03
SOMETIMES I CAN WHISTLE THROUGH BOTH NOSTRILS. I'VE SAVED A FORTUNE IN HARMONICAS.

- Pair programming doesn’t work well in our environment.
  - We’ve had some limited success, but most of the time, it’s not something we practice.
  - Our experience is neither unique nor universal: there are many people who are productive in a pair programming setting.

Scrum: an Example Agile Implementation

- Collaborate with the users to create user stories.
- Assign effort (points) to each user story.
- Prioritize the stories based on user priority.
- For each iteration:
  - Pick a few stories to deliver
  - Design, code, test
  - If something wasn't finished, reprioritize it (usually for the next iteration).
  - Get user feedback on the delivered functionality.
  - Have a retrospective about the iteration: what worked well? What were the problems? What can be improved?
- Meet daily ("Stand up meeting") to briefly discuss progress.
- As iterations progress, user stories might be added, changed or removed.

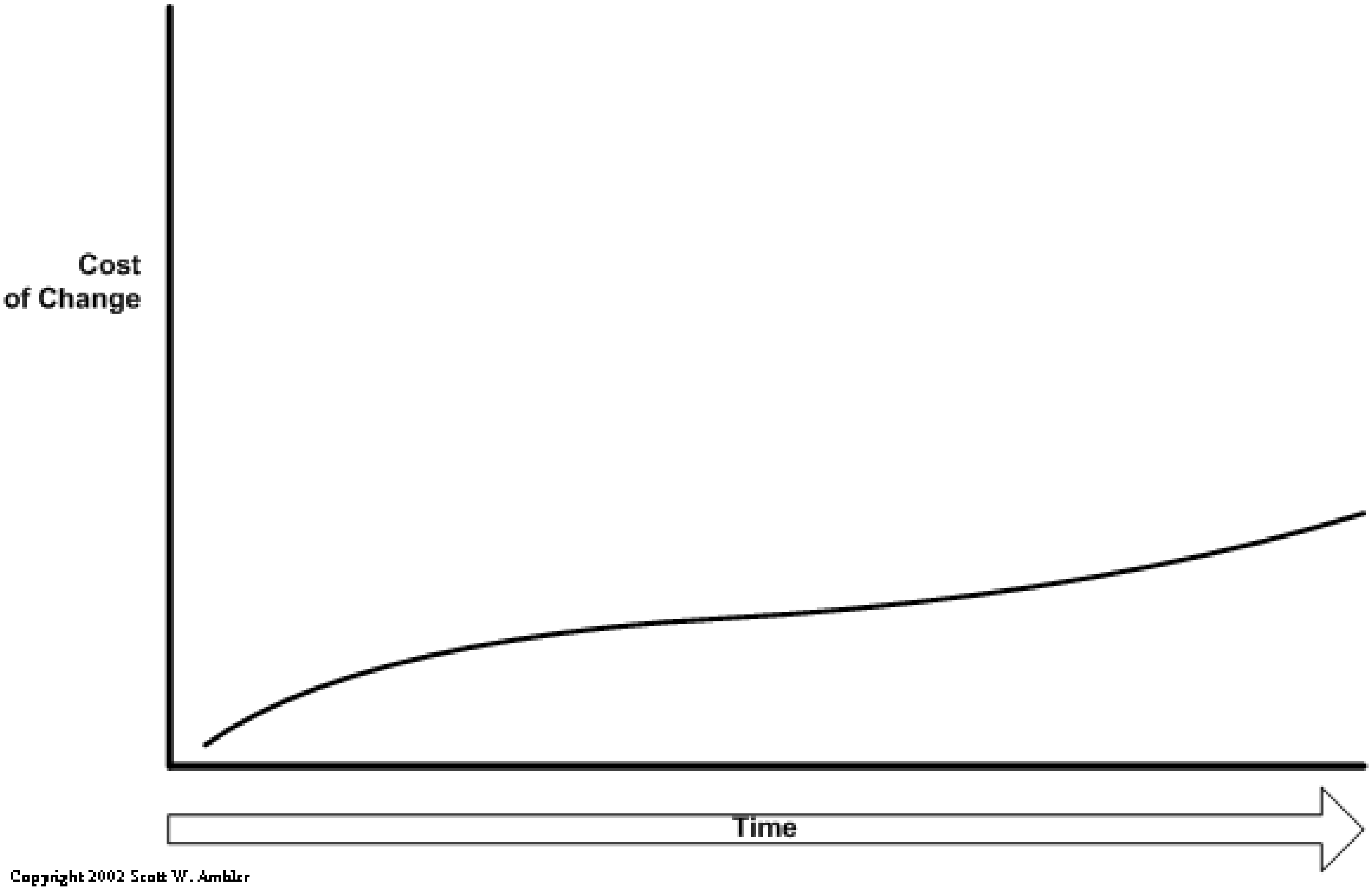
User Stories

- A user story captures requirements from the user’s perspective.
- They are typically short (one sentence) descriptions.
  - Template: “As a **<role>**, I want **<goal>** because **<reason>.”**
- They are purposely not detailed with implementation level descriptions.
- User stories are assigned effort estimates.
  - Some people like time units (day, ideal day, hour).
  - Time units can be confusing, so many practitioners have switched to using unit-less points (“story points”).
- Estimates are not meant to be super accurate. It’s often sufficient to group stories into a few categories (“easy”, “medium”, “hard”) and assign a set value to each category.
  - Tip: use an exponential scale, like “1, 3, 9” or “1, 4, 16”. The relationship between difficulty and effort is often non-linear.

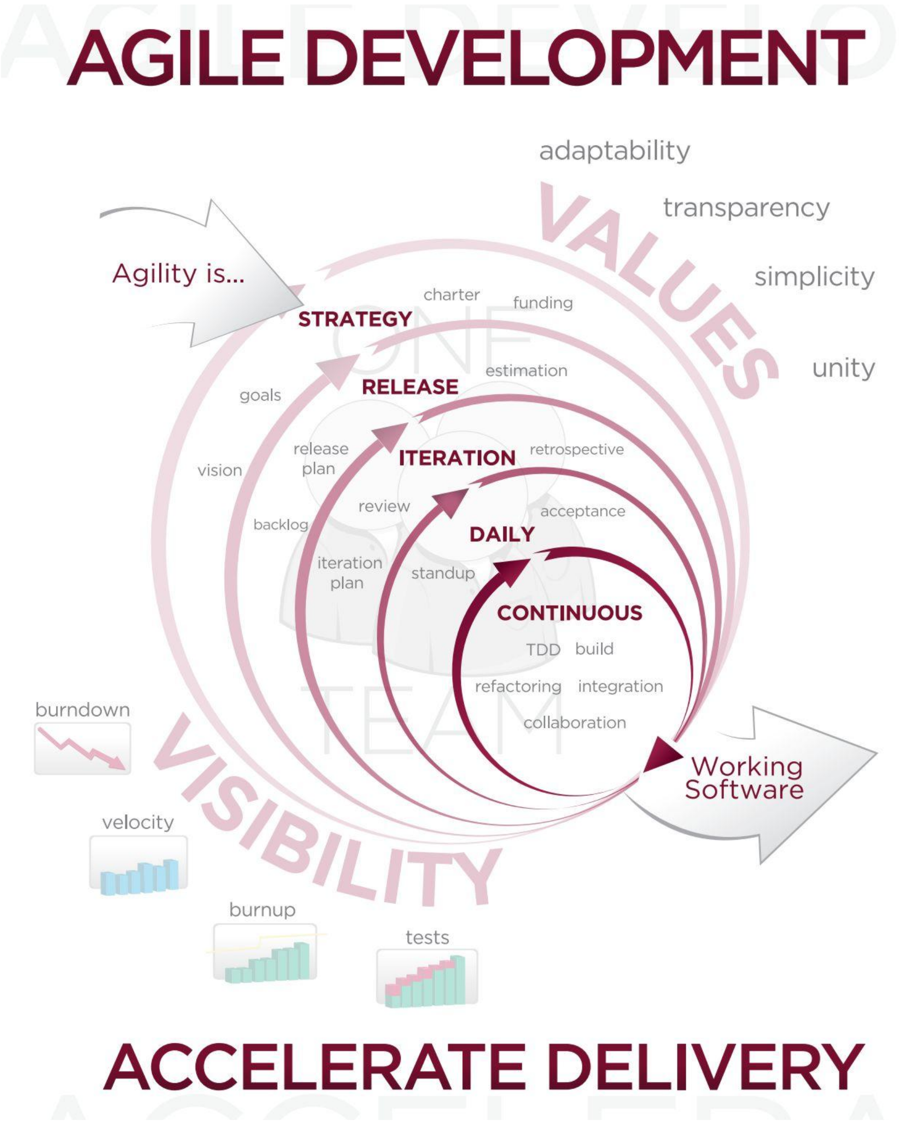
Iteration Planning and Velocity

- For each iteration, the team decides what stories they will deliver, taking into account user priorities and team velocity.
- The team’s velocity is defined as the number of story points delivered in each iteration.
- Velocity can fluctuate, but barring significant changes, the team’s recent velocity is a good indication of their delivery capacity for the near future.
- Real world advice:
  - Avoid the temptation to measure individual velocity.
  - For the first iteration, don’t worry too much about velocity. The first iteration requires some ramp up time, so purposely choose fewer story points than you think you can deliver.

Agile Cost of Change



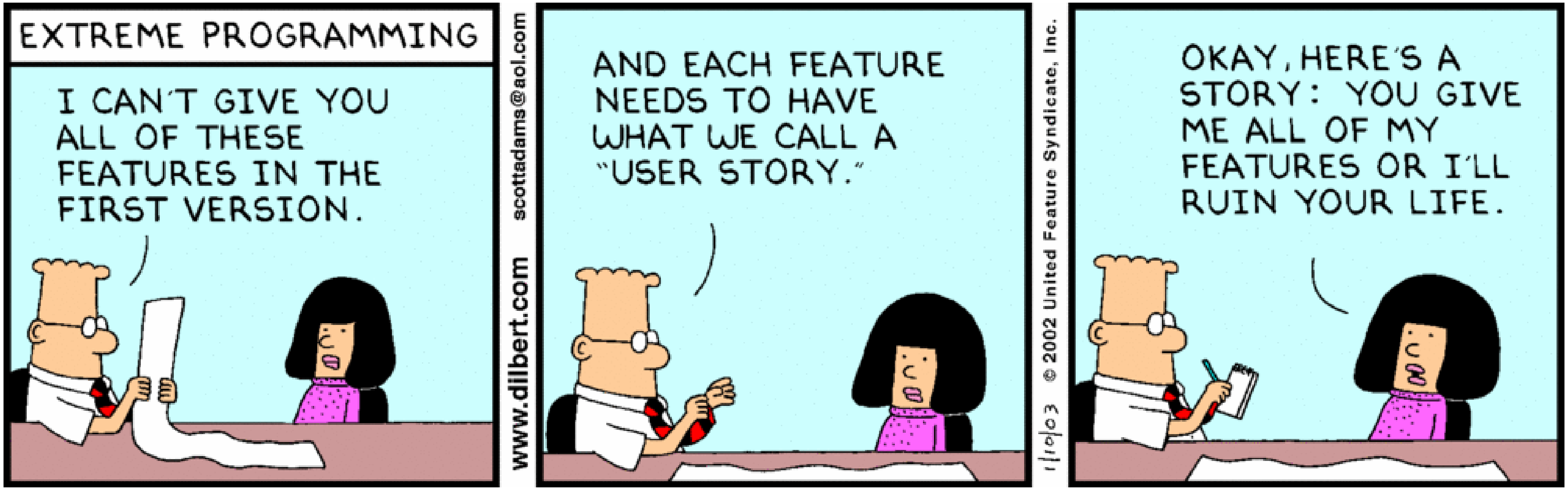
Agile Visualized



(From Wikipedia)

Agile Failure Scenarios

- No customer buy in.
  - Lack of executive support.
  - Lack of user participation.
- Bad organizational fit.
- Heart-lung machine, Flight Avionics.


EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
www.dilbert.com scottadams@aol.com
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
1/10/03 © 2002 United Feature Syndicate, Inc.
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.

Conclusion & Closing Thoughts

- Building software is very different than building a bridge, a skyscraper or a cruise ship.
- Agile as a Software Engineering paradigm emphasizes short feedback loops coupled with doing only what is necessary
- Agile works well in scenarios where change is inevitable. Most software development falls into that category.
- Agile technical practices reduce the cost of change over time.
- The management practices foster collaboration and reduce the distance between the vision and the implementation.
- Caveat emptor: process is not a substitute for wit and passion.