The Eclipse Way

Daniel Megert

Eclipse Platform and JDT Lead

IBM Research - Zurich

Why Did We Do Eclipse?

- Disrupt the growing dominance of Microsoft
- Solve our tool integration problems
- Create a community of plug-in providers

How we Started: Closed development

- The Swiss Bank approach to software development
  - If it hasn't shipped it doesn’t exist
- Strong firewall between developers and customers

History

- **1998 - IBM conceives idea of universal tool integration platform**
  - Work starts on SWT
- **1999 - IBM team starts work to build Eclipse Platform and Java IDE**
  - Based on 10 years experience with Smaltalk, VA/Java, VA/MicroEdition
- **2001 - IBM donates Eclipse Platform and Java IDE to open source ($40M)**
- **2001 - IBM Eclipse team leads Eclipse evangelism and seeds community**
  - IBM funded receptions and Eclipse community events
  - Keynotes, conference talks, articles by IBM technical leaders
  - 55 Full time developers improving Eclipse and fully engaged with community
  - First Eclipse-based products

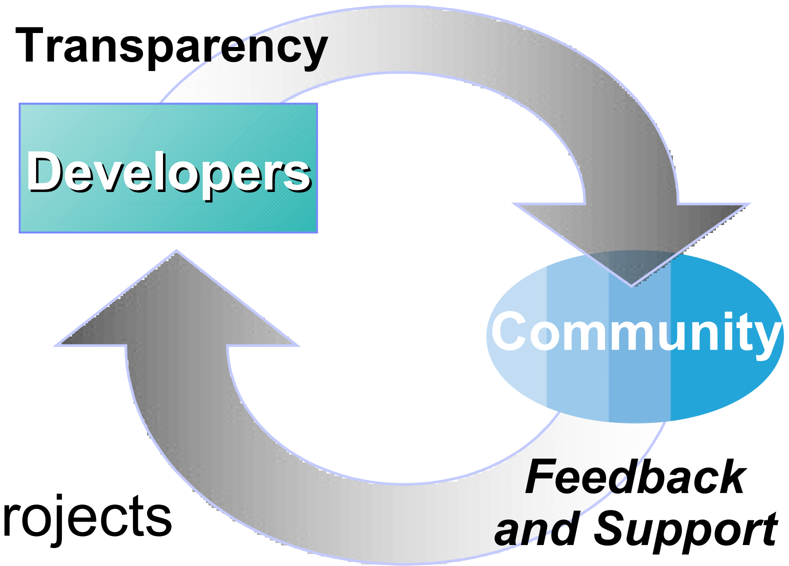
November 2001: "Open Source"

Reaction from the development team

You want us to do what?
Why are we doing this again?
Code in public?
Answer all those dumb questions?
Have technical discussions in public?

Key Lessons

- Transparency helps existing development
  - Better understanding of current status
  - Responding to feedback takes time, but pays off
- Use same communication channels inside as outside



➢ Not limited to Open Source projects
  ➢ "Open Commercial Development"

The Eclipse Way

- The secret of the success of the Eclipse team
- An agile software development process
- Used, developed and improved over time by the Eclipse team

The Success of the Process

- **The Eclipse team is shipping high quality software on-time for many years now**
  - Continuous nightly builds on-time
  - Weekly integration builds on-time
  - Six week milestones on-time
  - Yearly releases on-time
  - Service releases on-time

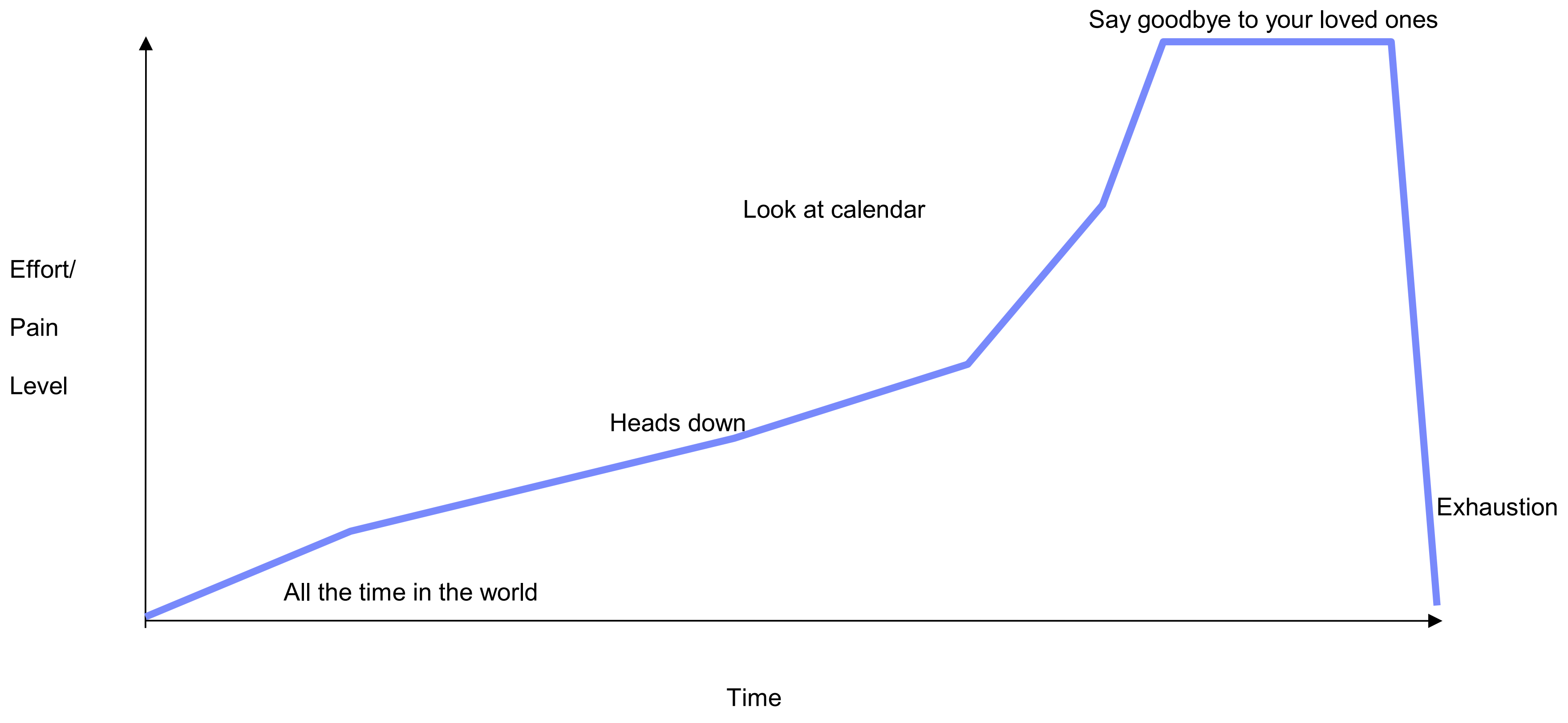
- **A healthy project**
  - Works on this high-level over years
  - Continuously improving the process

Eclipse 1.0	Nov 2001
Eclipse 2.0	June 2002
Eclipse 2.0.1	Sept 2002
Eclipse 2.0.2	Nov 2002
Eclipse 2.0.3	Mar 2003
Eclipse 2.1	Mar 2003
Eclipse 2.1.1	June 2003
…	…
Eclipse 4.5.1	Sept 2015

Getting Started

- **Milestones first**
  - Small cycles (+/- six weeks)
- **Early incremental planning**
  - Essential for many agile processes
- **Continuous testing, Continuous integration**
  - Essential for many agile processes
- **Endgame**
  - Stabilizing the product at the end of the release cycle
  - No feature work allowed
- **Decompression**
  - Essential to recover and improve the process over time

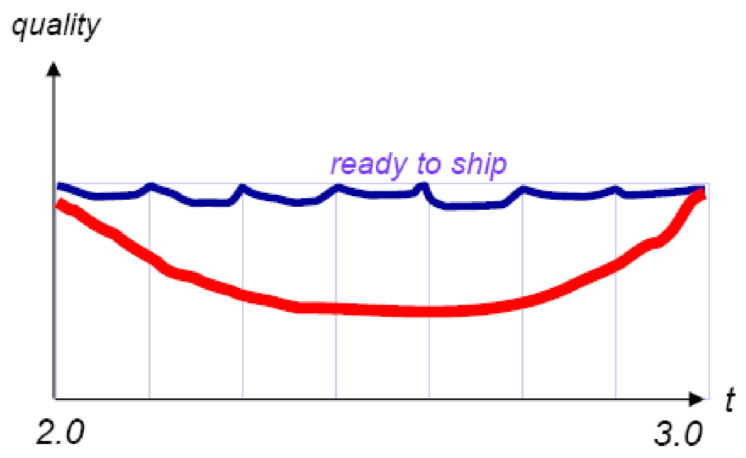
In the Past…


Say goodbye to your loved ones
Look at calendar
Effort/
Pain
Level
Heads down
Exhaustion
All the time in the world
Time

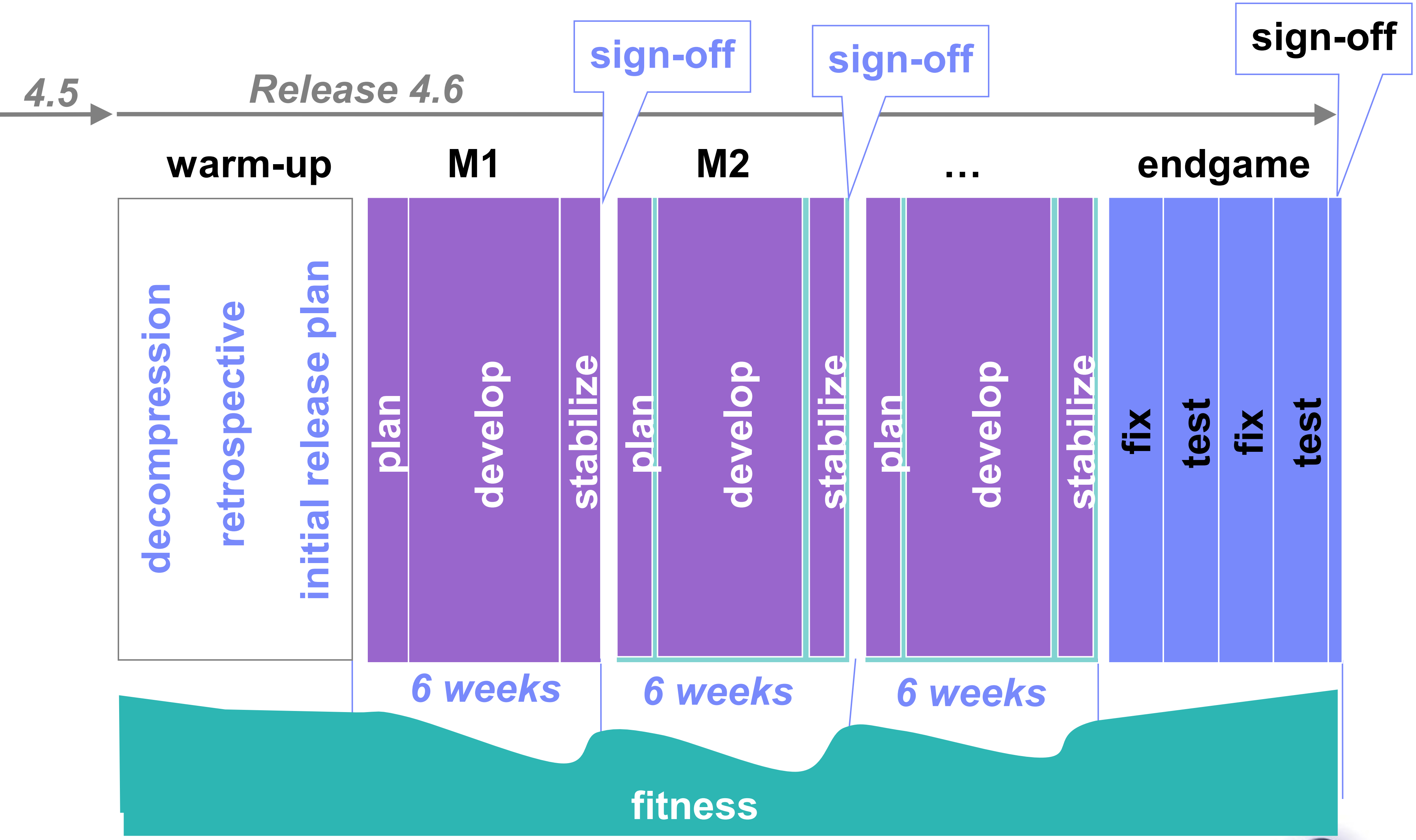
Milestones First

- **Break down release cycle into milestones**
  - We currently use 6 weeks
- **Each milestone is a miniature development cycle**
  - Plan, execute, test
  - Teams refer to the release plan when creating milestone plans
    - Assign plan items to a milestone
    - Milestone plans are public
- **Result of a milestone**
  - Milestone builds: good enough to be used by the community
- **Milestones reduce stress!**

- before/after



Iterative – Time-boxed



Early Planning

- Release themes **establish big picture**
  - Team input
  - Community input
- **Component teams define component plans**
- **PMC collates initial draft project plan**
  - Tradeoff: requirements vs. available resources
  - Committed, Proposed, Deferred

The Plan is Alive

- **The project plan is updated quarterly to reflect**
  - Progress on items
  - New items
  - Input from the community
- **Becomes final at the end of the release**
- **Before, and still practiced by many: static plans**
  - Accurate once, but no early feedback: non-existent until late in the cycle.

Continuous Integration

- **Fully automated build process**
- **Build quality verified by automatic unit tests**
- **Staged builds**
  - Nightly builds
    - Discover integration problems between components
  - Weekly integration builds
    - All automatic unit tests must be successful
    - Good enough for our own use
  - Milestone builds
    - Good enough for the community to use

Always Beta

- **Each integration build is a release candidate; we expect it to work**
- **Results of the build process and the automatic tests**
  - Indicate where we are
- **As tool makers we use our own tools**
  - Component teams use weekly integration builds
  - Community uses release and milesone builds
- **Continuously *Consume Our Own Output***
  aka Eat your own dog food

Community Involvement

- ***Problem*: no one knew what was in a milestone,**
  - So there was no incentive to move to milestone builds
  - So we received minimal feedback
    - More stale defect reports
  - Quality suffered
- ***Solution*: publish New and Noteworthy**
  - Advertise what we have been doing
- **Requires transparency**
  - Community needs to know what is going on to participate
- **Requires open participation**
  - We value the contributions of the community
- **We are the community**

Testing

- **Innovate and refactor code with confidence**
  - Continuous incremental design
- **Almost 90,000 JUnit tests**
- **Tightly integrated into the build process**
  - Tests run after each build (nightly, integration, milestone)
  - Milestone builds are only green when all tests pass
- **Test / Report kinds**
  - **Correctness** tests**:** Assert correct behavior
  - **Performance** tests**:** Allow to see performance regressions
    - Based on a database of previous test run measurements
  - **Resource** tests: no leaks and no resource consumption regressions
  - API verification - breakage
  - API verification - illegal use of internal/non API

Endgame

- **Convergence process applied before release**
  - Sequence of test-fix passes (RCs)
    - Community event
- **With each pass the costs for fixing are increased**
  - Higher burden to release a fix for a problem
  - Focus on higher priority problems and trivial fix/polish items
- **Endgame endurance**
  - We are only effective for so long
  - Distribute Quality/Polish effort throughout the release
  - Shared responsibility and commitment
  - We all sign off

Decompression

- **Recover from release**
- **Retrospective of the last cycle**
  - Achievements
  - Failures
  - Process
  - Cross-team collaboration
- **Explore new stuff**
- **Start to plan the next release and cycles**

Conclusion

- The team makes the process work
- The team defines and evolves the process

eclipsecon Europe

Ludwigsburg, Germany, 3 - 5 November 2015

Evaluate the sessions at www.eclipsecon.org

+1 0 -1