agile, open source, distributed, and on-time

inside the eclipse development process

Erich Gamma
IBM Distinguished Engineer
Eclipse Project Management Committee
erich_gamma@ch.ibm.com

© 2005 IBM Corporation

what is eclipse (www.eclipse.org)?

- an IDE and more…
  - it's a Java development environment (JDT)
  - it's a general tools and integration platform
  - it's a general application platform (RCP)
- an open source community
- an ecosystem to enable a total solution
  - including products by some major tool vendors
- a foundation to advance the eclipse platform

© 2005 IBM Corporation

inside the eclipse development process

- our process has evolved over time
  - stay aware, adapt, change
- our process has helped us to achieve:
  - predictability
  - quality delivery on time
- ➢ share what we have learned
  - ➢ influence “by example” – similar as the eclipse Java Development Tools

© 2005 IBM Corporation

what it isn’t…

- shipping software is not about...
  - having the right development environment
    - we know, we build IDEs
  - having the right methodology

- software engineering involves all these...

- ...but we’ll bet on the people every time!

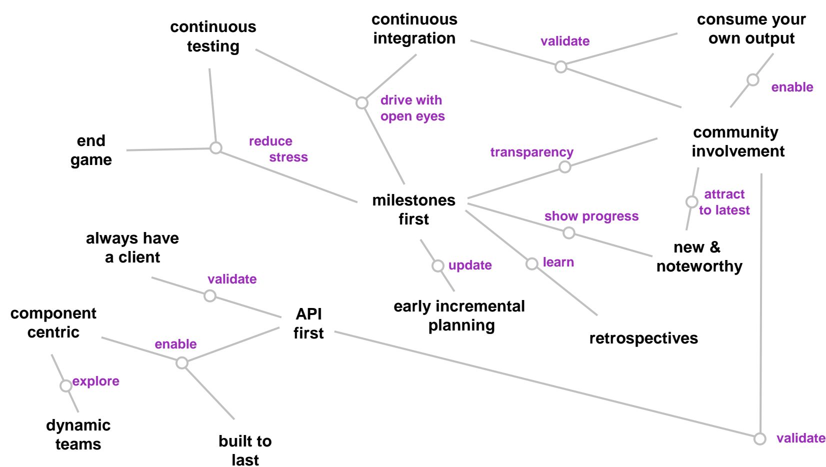
© 2005 IBM Corporation

“*we ship software*"

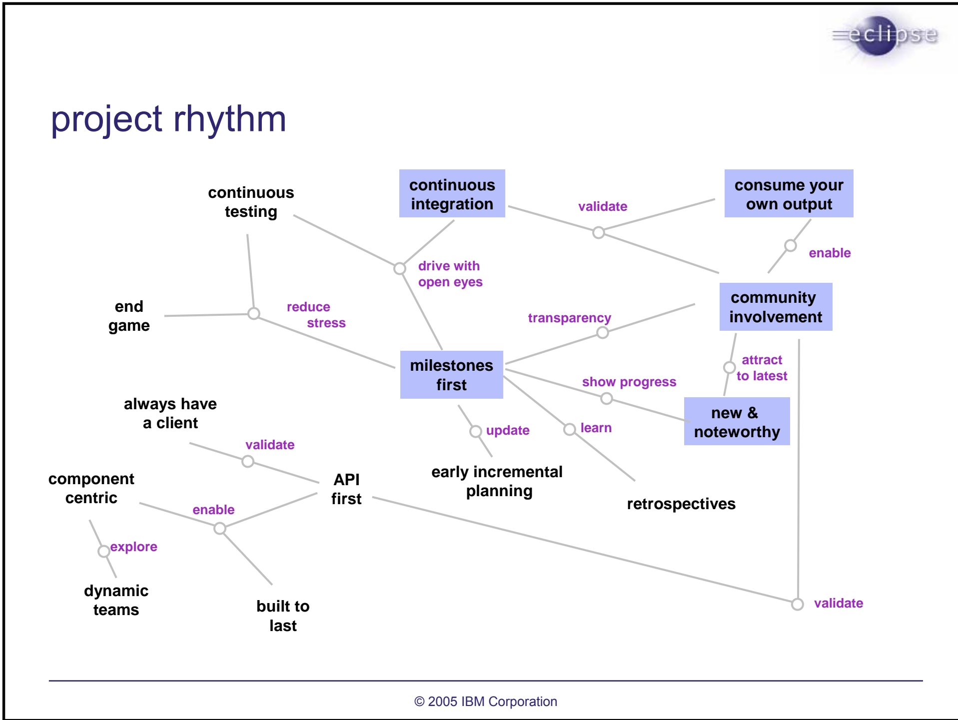
- our objective is to ship software
  - anything that contributes to this goal is good
  - anything that does not is bad
- team member recognition is based on the ability to ship
  - quality software
  - on time
- the culture: "*If you ship, then you may speak.*"
- the question: "*Did they ever ship anything?*"
- the insult: "*They never ship anything!*"

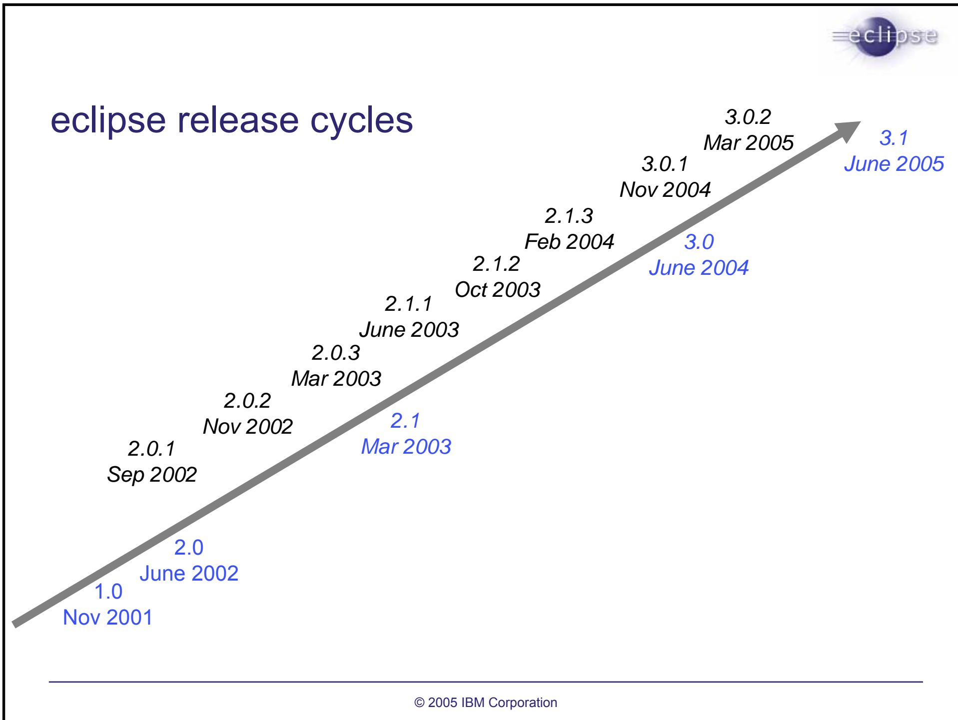
© 2005 IBM Corporation

eclipse practices



© 2005 IBM Corporation


project rhythm
continuous testing
continuous integration
validate
consume your own output
drive with open eyes
enable
end game
reduce stress
transparency
community involvement
milestones first
attract to latest
show progress
always have a client
update
learn
new & noteworthy
validate
early incremental planning
component centric
API first
retrospectives
enable
explore
dynamic teams
built to last
validate
© 2005 IBM Corporation

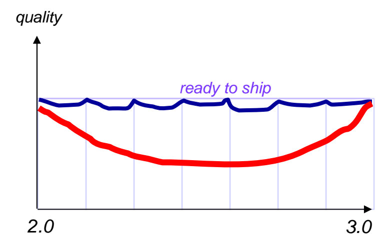

eclipse release cycles
3.0.2 Mar 2005
3.1 June 2005
3.0.1 Nov 2004
2.1.3 Feb 2004
3.0 June 2004
2.1.2 Oct 2003
2.1.1 June 2003
2.0.3 Mar 2003
2.0.2 Nov 2002
2.1 Mar 2003
2.0.1 Sep 2002
2.0 June 2002
1.0 Nov 2001
© 2005 IBM Corporation

milestones first

- break down release cycle into milestones
  - we currently use 6 weeks
- each milestone is a miniature development cycle
  - plan, execute, test, retrospective
  - 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
- at the end of each milestone we do a retrospective
  - what went well, what did not?

➢ milestones reduce stress!

- before/after



"hanging rope"

© 2005 IBM Corporation

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
- **reality**: build failures occur
  - rebuild to create acceptable integration, milestone builds.

© 2005 IBM Corporation

always beta

- each 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 team – use the latest code daily
  - project team – use integration builds (weekly)
  - community – use milestone builds
- ➢ continuously *Consume Our Own Output*

© 2005 IBM Corporation

it is a community thing

- software development (Open Source even more) is a "community thing"
  - an active community is the major asset of an OS project
- OS project gives and takes:
  - OS developer gives:
    - listen to feedback and react
    - demonstrate continuous progress
    - transparent development
  - OS developer takes:
    - answer user questions so that developers do not have to do it
    - report defects and feature requests
    - validate technology by writing plug-ins
    - submit patches and enhancements
- Give and take isn't always balanced
  - community isn't shy and is demanding

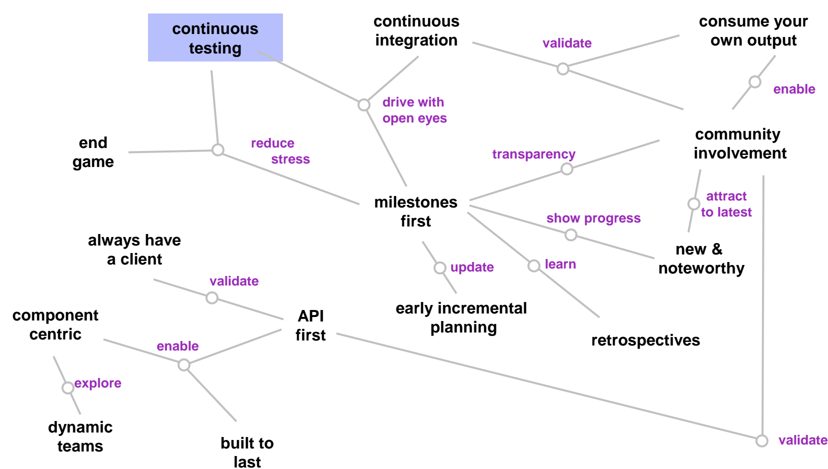
© 2005 IBM Corporation

community involvement

- 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
- *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

© 2005 IBM Corporation

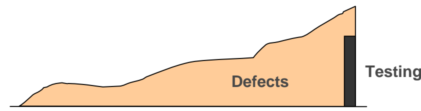
testing

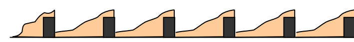


© 2005 IBM Corporation

testing

- innovate with confidence
  - continuous incremental design
- > 20,000 JUnit tests
  - tightly integrated into the build process
    - tests run after each build (nightly, integration, milestone)
  - integration builds are only green when all tests pass
- test kinds
  - **correctness** tests
    - assert correct behavior
  - **performance** tests
    - assert no performance regressions
      - based on a database of previous test run measurements
  - **resource** tests, leak tests
    - assert no resource consumption regressions





Kent Beck – JUnit handbook

© 2005 IBM Corporation

test report

Eclipse SDK

The Eclipse SDK includes the Eclipse Platform, Java de
aren't sure which download you want... then you probabl
Eclipse does not include a Java runtime environme
run Eclipse. Click here if you need help finding a Java ru

Status	Platform
✔	Windows 98/ME/2000/XP
✔	Linux (x86/Motif) (Supported Versions)
X	Linux (x86/GTK 2) (Supported Versions)
✔	Linux (AMD 64/GTK 2) (Supported Versions
✔	Solaris 8 (SPARC/Motif)
✔	AIX (PPC/Motif)
✔	HP-UX (HP9000/Motif)
✔	Mac OSX (Mac/Carbon) (Supported Versions)
✔	Source Build (Source in .zip) (instructions)
✔	Source Build (Source fetched via C

Unit Test Results

Summary

Tests	Failures	Errors	Success rate	Time
352	1	0	99.72%	287.698

Packages

Name	Tests	Errors	Failures	Time(s)
org.eclipse.jdt.debug.tests	352	0	1	287.698

Package org.eclipse.jdt.debug.tests

Name	Tests	Errors	Failures	Time(s)
AutomatedSuite	352	0	1	287.698

© 2005 IBM Corporation

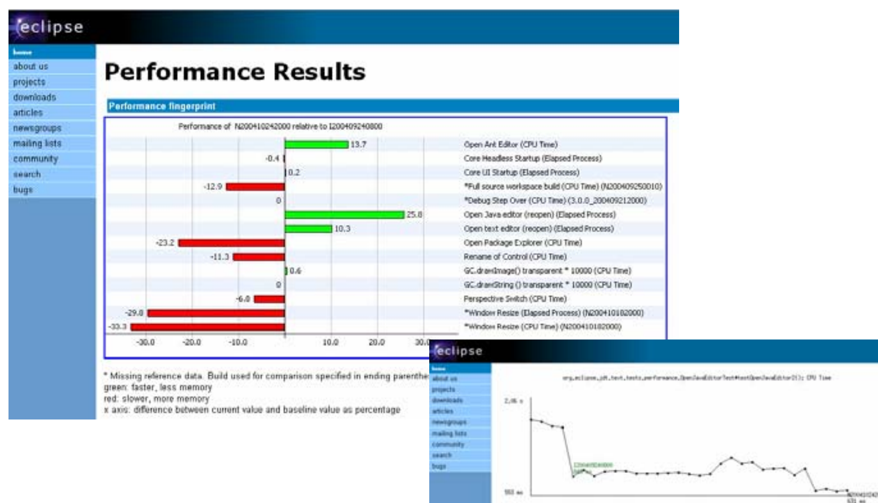
performance testing

- performance tests are an ongoing part of Eclipse 3.1
  - keep closer tabs on performance changes during development
- JUnit performance test

```
public class MyPerformanceTestCase extends PeformanceTestCase {
  public void testMyOperation() {
    startMeasuring();
    for (int i= 0; i < 10; i++) {
      toMeasure();
    }
    stopMeasuring();
    commitMeasurements();
    assertPerformance();
  }
}
```

© 2005 IBM Corporation

performance test report



© 2005 IBM Corporation

before (M5) – after (M7)

Performance of I20050219-1500 relative to 3.0

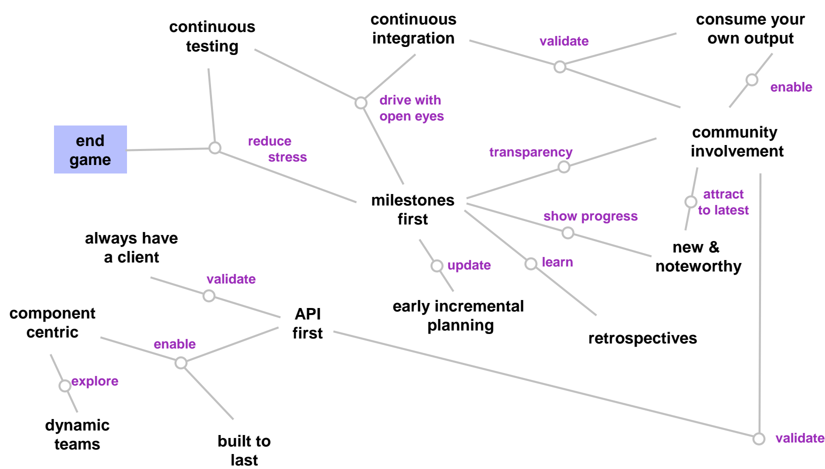
Win XP Sun 1.4.2_06

Performance of 3.1M7 relative to 3.0

Win XP Sun 1.4.2_08 (3 GHz 2 GB)

© 2005 IBM Corporation

eclipse practices



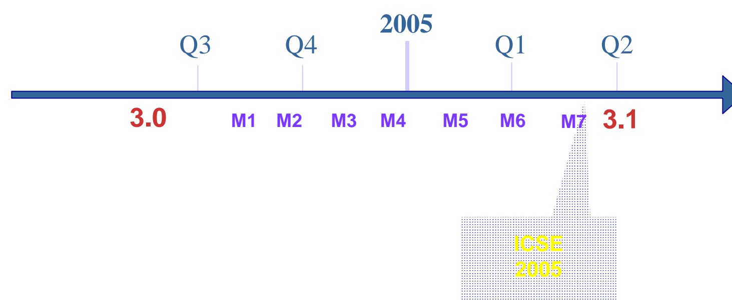
© 2005 IBM Corporation

endgame

- endgame endurance
  - we are only effective for so long
  - distribute quality/polish effort throughout the release
- shared responsibility and commitment
  - we all sign off

© 2005 IBM Corporation

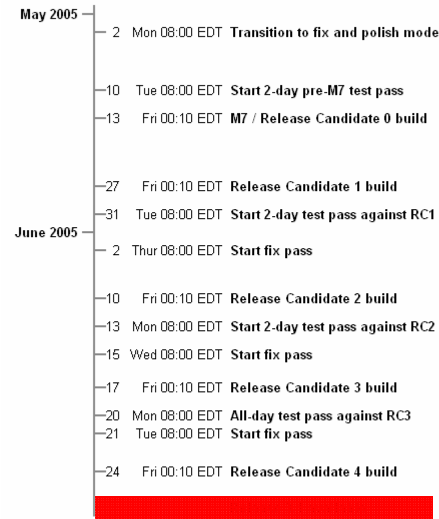
where are in the middle of the 3.1 end game



© 2005 IBM Corporation

3.1 endgame

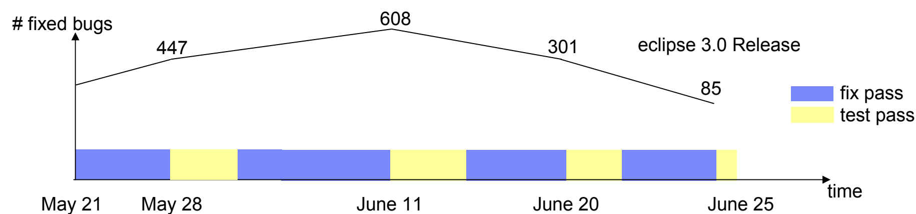
- convergence process applied before release
- sequence of test-fix passes
  - it is a community event!



© 2005 IBM Corporation

endgame convergence

- with each pass the costs for fixing are increased
  - higher burden to work on fix for a problem
  - higher burden to release a fix for a problem
  - focus on higher priority problems



© 2005 IBM Corporation

signing-off

- "An enthusiastic GO from PDE" - Cherie
- "The best build of the year!" - Dejan
- "An Eclipse build that the whole family can enjoy" - Wassim

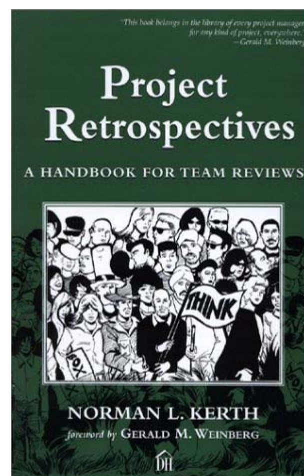
- Your PDE team.

© 2005 IBM Corporation

decompression

- recover from release
- retrospective of the last cycle
  - learn from the last cycle
    - achievements
    - failures
  - “stay aware, adapt, change”
  - define retrospective actions

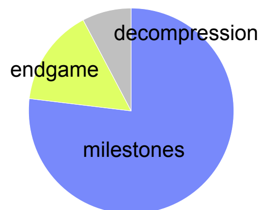
- start to plan the next release and cycle



© 2005 IBM Corporation

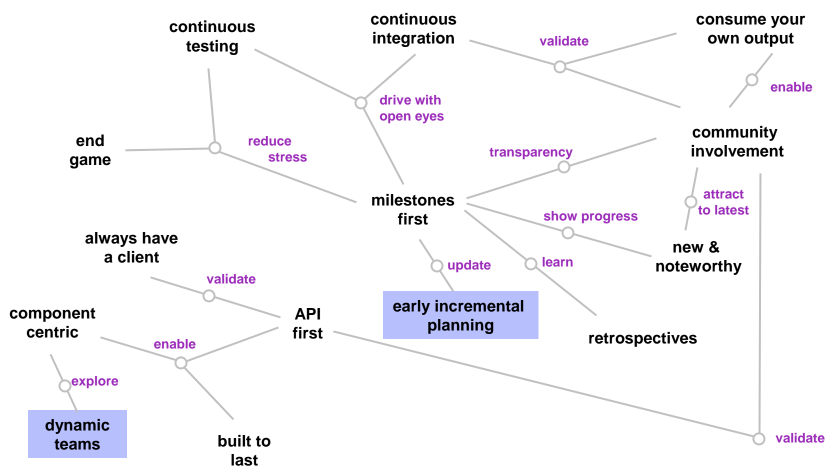
where the time goes

- release cycle 12-16 months
  - milestones - 9 months
  - endgame – 1-2 months
  - decompression – 1 month



© 2005 IBM Corporation

planning



© 2005 IBM Corporation

early planning

- release themes establish big picture
  - community input
  - requirements council new source of input
- component teams define component plans
- PMC collates initial project plan draft
  - tradeoff: requirements vs. available resources
  - committed, proposed, deferred
- plan initially spells out
  - themes
  - milestones
  - compatibility (contract, binary, source, workspace)

© 2005 IBM Corporation

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: static plans
  - accurate at one point in time only
  - but no early feedback: non-existent until late in the cycle.

© 2005 IBM Corporation

ongoing risk assessment

- address high risk items and items with many dependencies early
- maintain schedule by dropping items (if necessary)
  - we will drop proposed items
  - we hate to drop committed items
  - prefer fewer completed items than more items in progress
- high risk items are sandboxed to reduce risk to other items
  - prefer to serialize highest risk items (to minimize integration pain)

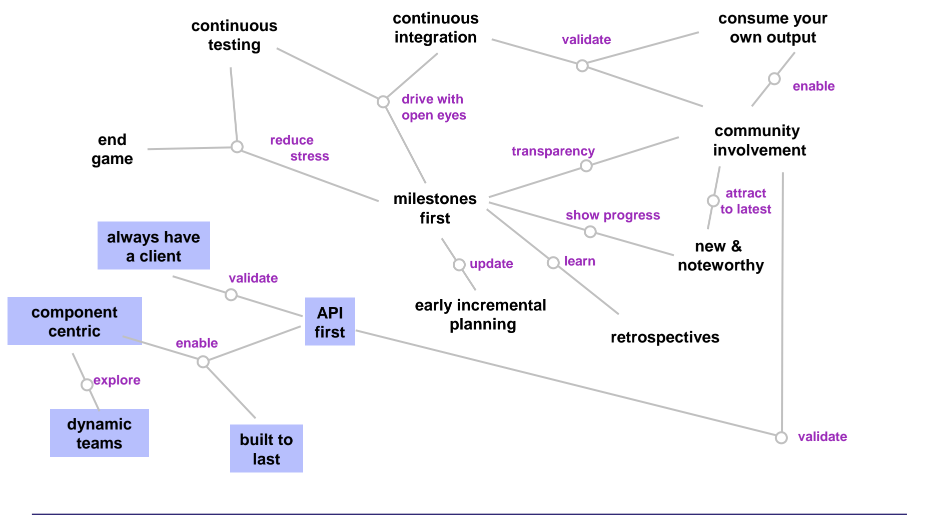
© 2005 IBM Corporation

collective ownership

- PMC meets at least once a week
- all component leads and the PMC meet for a weekly planning call
  - status
  - planning
  - identification of cross-component issues
  - meeting notes posted to the developer mailing lists
- dynamic teams are established for solving cross-component issues
  - one cross-component issue per dynamic team
  - members are key developers from all effected components
  - find, implement, and roll-out solution of the assigned cross component issue
  - represented in the weekly planning calls

© 2005 IBM Corporation

built to last



© 2005 IBM Corporation

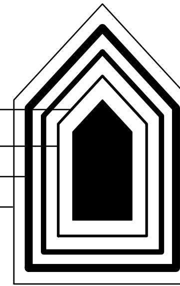
built to last

- deliver on time, every time
  - decisions in this release impact what we can do next release
  - must preserve architectural integrity
- deliver quality
  - innovation with continuity
  - need to have a solid foundation
    - scalable
    - performant
    - stable

© 2005 IBM Corporation

how buildings last

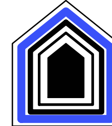
- **Stewart Brand: how buildings learn – what happens after they're built**
  - stuff: furniture
  - services: electrical, plumbing (7-15y)
  - structure: foundation, load bearing walls (30-300y)
  - site: geographical setting (forever)



- **layers:**
  - evolve at different rates during the life of a building
  - shear against each other as they change at different rates
  - an adaptive building must allow slippage
  - ➢ a building that lasts is adaptive and can change over time
  - ➢ lasts for generations without total rebuilding

© 2005 IBM Corporation

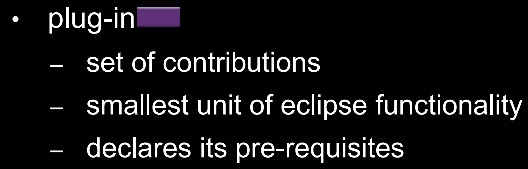
structure foundation



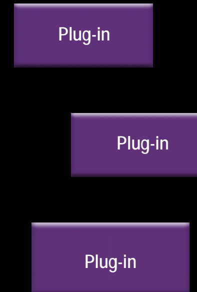
- the eclipse plug-in architecture
- everything is a plug-in
  - simple and consistent

© 2005 IBM Corporation

everything is a plug-in

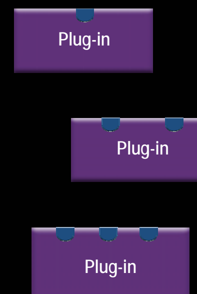


- plug-in
  - set of contributions
  - smallest unit of eclipse functionality
  - declares its pre-requisites



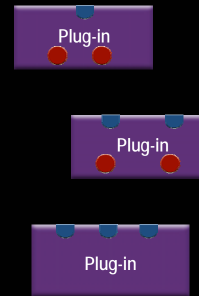
everything is a plug-In

- extension point
  - named entity for collecting contributions



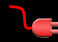
everything is a plug-in

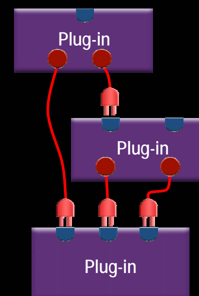
- extension
  - a contribution



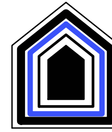
everything is a plug-in

- the platform run-time is making the connection





services plumbing: APIs



- APIs matter
  - define consistent, concise API
    - define API conventions (*.internal.* in eclipse)
  - don't expose the implementation
  - develop implementation and client at the same time
- define APIs for stability
  - binary compatibility is highest priority
  - we would rather provide less API than desired (and augment) than provide the wrong (or unnecessary) API and need to support it indefinitely

© 2005 IBM Corporation

component centric

- component centric development
  - a team is responsible for one or more component
- dependencies through APIs
  - ensures high velocity development inside a component
  - *eclipse 3.1 provides tools support to check for API access violations*
    - *as you type*
- define producer/consumer relationships among components
  - tension among components is healthy for coming up with good component interfaces/APIs

© 2005 IBM Corporation

APIs first

- APIs don't just happen; we need to design them
- specifications with precisely defined behavior
  - what you can assume (and what you cannot)
  - it works ≠ API compliant
  - documented classes ≠ API
  - "provisional API" = API that didn't make it for a release
- must have at least one client involved, preferably more
- need an API advocate
  - we all care about having sustainable APIs
  - need someone who lives and breathes APIs

© 2005 IBM Corporation

practical extensibility

- extensible in ways that are known useful
  - needed by us, requested by others
- we don't provide hypothetical generality - we want to be extensible in ways that matter
  - don't over generalize

© 2005 IBM Corporation

API tension

- conflict
  - APIs need to be implemented and used to be right
    - requires iteration
    - needs external client
  - stable API necessary
    - for widespread adoption
- resolution
  - don't commit API before its time
    - APIs can (and should) change during the release to accommodate new requirements, experience

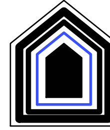
© 2005 IBM Corporation

example: API evolution in the Java Development Tools

- new APIs
  - AST (Abstract Syntax Tree)
  - AST rewriting
    - code manipulation
- open-up
  - contribute to quick fix/quick assist
  - contribute to code assist
- push-down
  - make JDT specific support available to other languages:
    - template processors
    - linked editing

© 2005 IBM Corporation

stuff, furniture - UI



- eclipse extension architecture is contribution based
  - extensions contribute to the workbench
  - the workbench manages and presents the contributions
- enables UI evolution
  - 3.0 new look

© 2005 IBM Corporation

keeping the house clean…

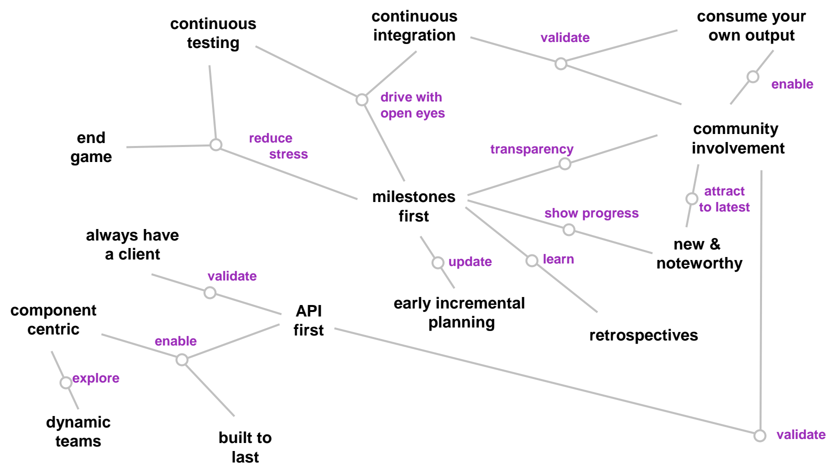
- blur the boundary between the Platform and contributions
  - plug-ins should fit in (and together) naturally
  - ➢ this is a shared responsibility for all plug-in developers

housekeeping rules

- **sharing rule**: Add, don't replace
- **invitation Rule**: Whenever possible, let others contribute to your contributions
- **fair Play Rule**: All clients play by the same rules, even me.
- **Relevance Rule**: Contribute only when you can successfully operate
- **Integration Rule**: Integrate and don't separate

© 2005 IBM Corporation

summary



© 2005 IBM Corporation

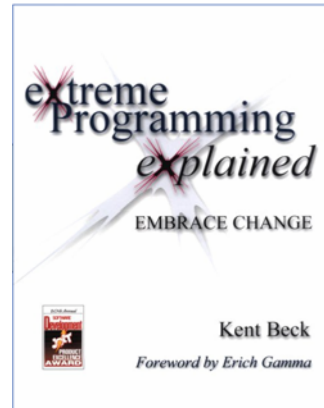
confessions

- ripples - a change in a lower layer that propagates up
  - we often underestimate the impact on upper layers!
    - consequence: end game swirls and stress
  - ground rule:
    - you are not done unless the upper most layer has digested the ripple
- dynamic teams – not all efforts are successful
  - require leadership, feeling of responsibility
  - require even more detailed planning
- we drop features!
  - but we hate to drop committed items

© 2005 IBM Corporation

summary agile practices followed

- ✓ *Testing early, often and automated*
- ✓ *Incremental design*
  - yes but API stability is important to us
- ✓ *Daily deployment*
- ✓ *Customer involvement*
  - we have an active community
- ✓ *Continuous integration*
  - nightly, weekly, milestone
- ✓ *Short development cycles*
  - 6 weeks cycles
- ✓ *Incremental planning*
  - time boxing
  - refined after each milestone



© 2005 IBM Corporation

conclusion

- the team makes the process work
- the team defines and evolves the process
- agile Open Source processes work
  - processes are applicable for closed source development

© 2005 IBM Corporation

So? (Your First Step)

- don't be afraid of Open Source products
  - join the Eclipse community (as one example…)

© 2005 IBM Corporation

So? (Your Next Steps)

- increase your aggressiveness
- consider "Community Source" development
  - use Open Source practices and tools for internal development
    - more transparent processes
    - more agile development
  - "OS can make a large organization act like a smaller one"
    - more control to the developers
    - information flows are nakedly visible
  - The OS development process is almost a literal translation of modern management principles.

© 2005 IBM Corporation

eclipse

So? (After All)

- consider Open Source development for your products
  - split your product into
    - commodity (platform) layer
    - differentiator (application) layers
      - keep the application layer small
  - become fully transparent
    - don’t hide your bugs and plans anymore
  - start to open source once you have something “interesting”
  - invest into building a community

© 2005 IBM Corporation