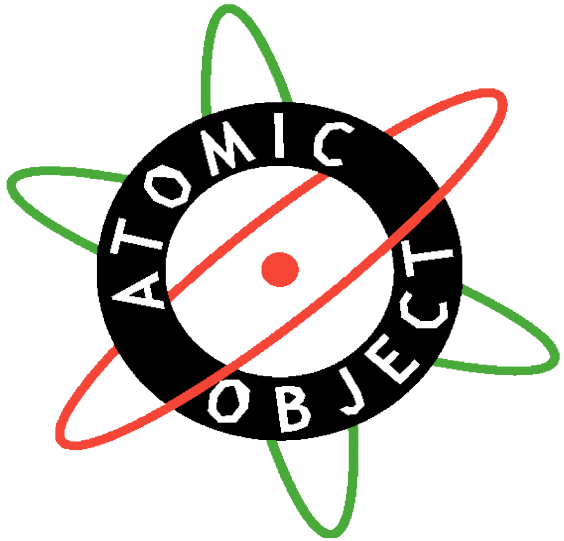
OO Testing: from academia to the real world

Software testing by small teams in a contract programming environment

Carl Erickson
Atomic Object LLC
Uppsala University

30 May 2002



Overview of this talk

- Summary of 1994 work on OO integration testing
- Atomic Object LLC testing methodology
- A hindsight perspective on my academic testing work

- Suggestions for improving our process

Unit testing

- The unit: single class or a method
- The functionality of methods, construction and destruction of objects
- At the unit testing level methods are procedural, so testing is the same
- Easy to generate test cases
- Readily automatable, suitable for regression testing

Integration testing

- Fewer obvious structural relationships to guide testing
- Design for reuse implies many possible compositions
- How we define a unit has an impact on integration testing

  (unit == method) => intraclass integration

  (unit == class) => interclass integration

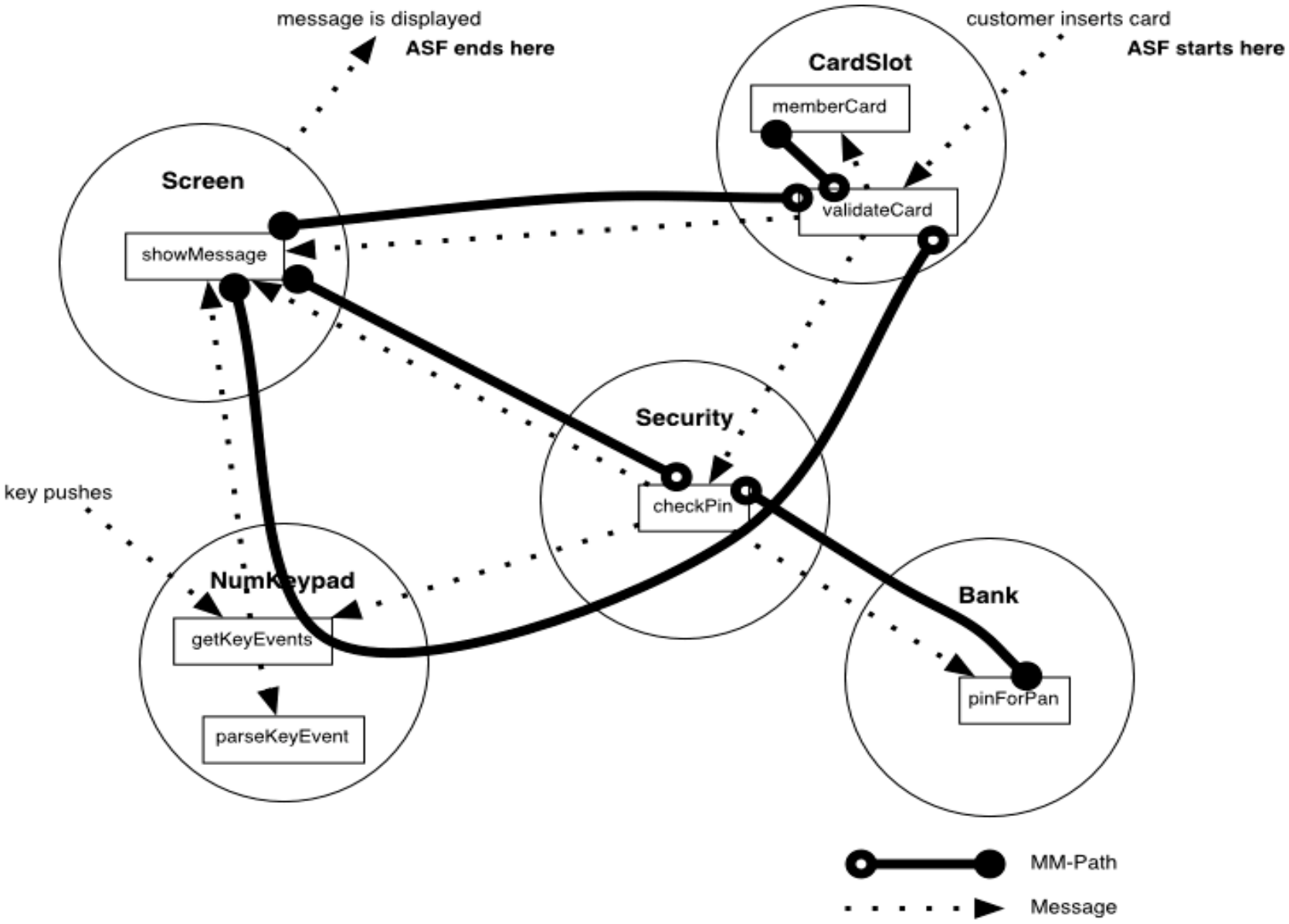
Integration testing (cont.)

- Some ideas for integration levels
  - Objects in the same package
  - Objects involved in a design pattern
  - Objects with relationships visible in class diagrams
- More difficult to generate test cases
- Generally automatable with some work
- Construct we proposed in ‘94: MM-Path

MM-Path

- Method Message-Path: sequence of method executions linked by messages
- MM-Paths may branch off from other MM-Paths
- End point is a method that issues no messages itself
- ATM example

Bankomat PIN entry use case



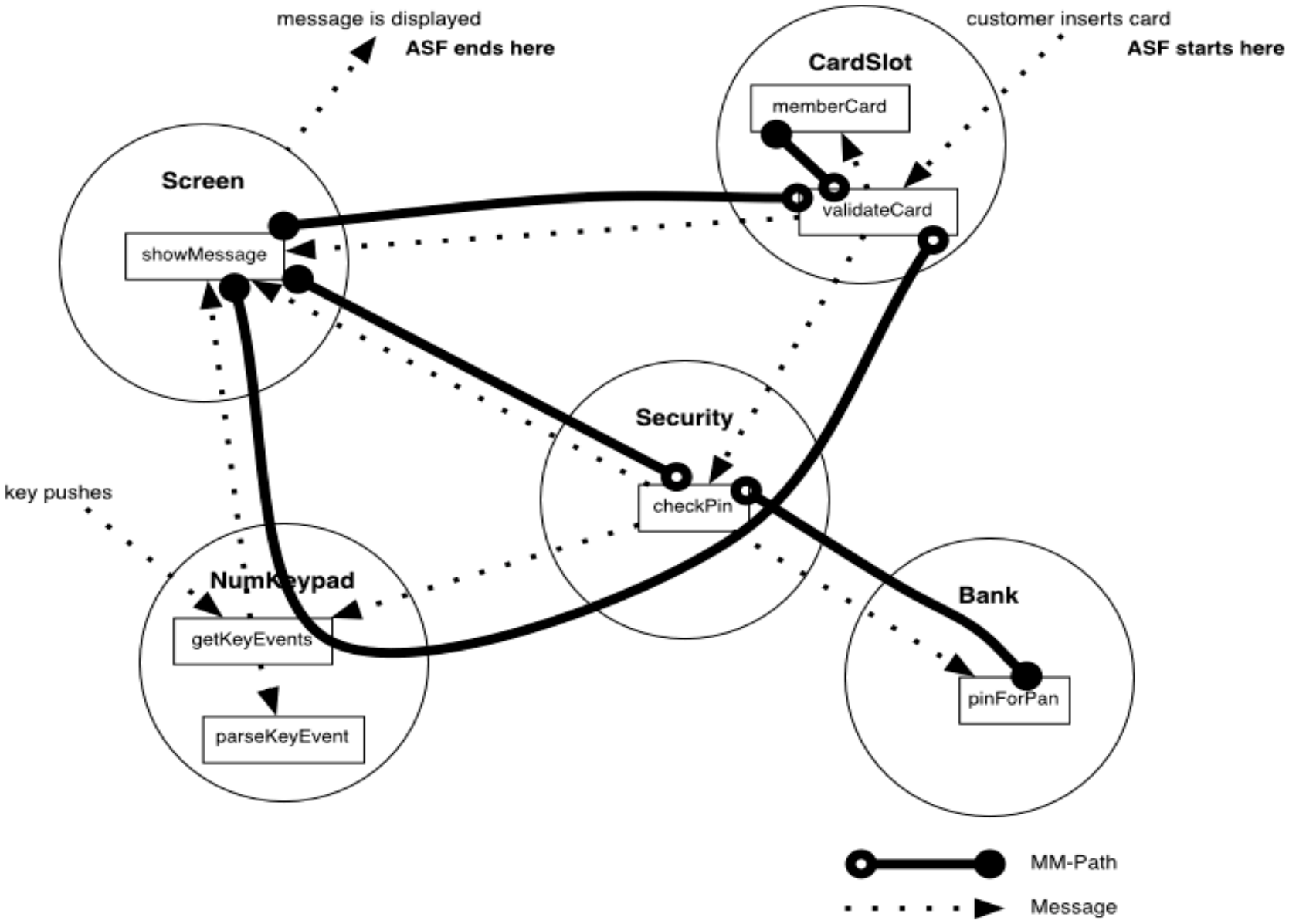
System testing

- Limited to events visible at the boundaries
- What a user can do with a system
- Correspond to user stories, use cases, acceptance tests
- Difficult to generate good test cases
- Very tough to automate (GUI problem)
- Construct we proposed in ‘94: ASF

Atomic System Function (ASF)

- Starts at an input port event
- Set of MM-Paths until an output port event is reached
- Often corresponds to something a user would do with the application

Bankomat PIN entry use case



Atomic Object testing methodology

- Testing as a marketing tool
  - Positive benefit of the “software crisis”?
  - Measuring and tracking the code base
- Test-first development
  - Simultaneous development of source and test
  - An idea from Extreme Programming
- Multiplatform testing
  - Improves code quality

Levels of testing

- What we test
  - each method thoroughly
  - higher level operations of each class (multiple methods)
  - operation of closely related classes
  - properties of patterns (e.g. singleton semantics)
  - entire subsystem functionality
- Conclusion
  - unit vs integration is a false dichotomy

System testing

- Testing use cases, user stories
  - Corresponds to ASFs
- How to build and release safely?
  - Current need driven by our first production release
  - Unit/integration test suites necessary but not sufficient
  - More than bugs: configuration problems
- GUIs make life difficult
  - Tedious manual process
  - Recent work with Robot
- An experiment
  - mode of operation of the application itself

Test-parallel development

- Quibble with the XP name (“test-first”)
  - Practical problems
- Testing as development methodology
  - Same people, same process, same time
- What good programmers do naturally
  - captured and preserved for lasting value
- Costs
  - Apparently higher initially
  - Quality and maintainability

What do we test?

- Test “everything that can break”
  - White-box, experienced-based, intuitive
  - Cardinal sin of missing tests
- Tests are devised in an ad-hoc fashion
  - MM-Paths for estimating coverage, identifying holes?
  - Automated tool possible?
- Subsystem tests are more methodical
  - All possible combinations of subsystem state
  - Distributed file system example

    Usage X Ownership X Storage X Connectivity

Automated regression testing

- Absolutely necessary to automate
- Test suites and xUnit framework
  - Composite pattern of tests
  - Setup, tear down, results
- Enables “fearless” development, continuous code improvement
- Lets anyone work on any code
- The higher the test, the harder to automate

Testing and integration

- All tests run 100% correct before you commit
- Limits the scope of problems, speeds up the process of finding them
- Less onerous to maintain continuously
- The system is always in a working state
  - Growing working complex systems
  - Great marketing
- Size of project determines implementation
  - 10 minute rule

Unexpected benefits of test-parallel development

- Requirements for a class are resolved earlier and more thoroughly
- Classes are more loosely coupled since they are designed to be testable in isolation
- Developers have tests as a form of class doc
- Pace of development is smoother, tests never fall behind code base, avoiding days in“testall hell”
- Get a small but positive psychological lift from tests passing 100%
- Minimize the anxiety of "releasing" incomplete or buggy code to your fellow developers

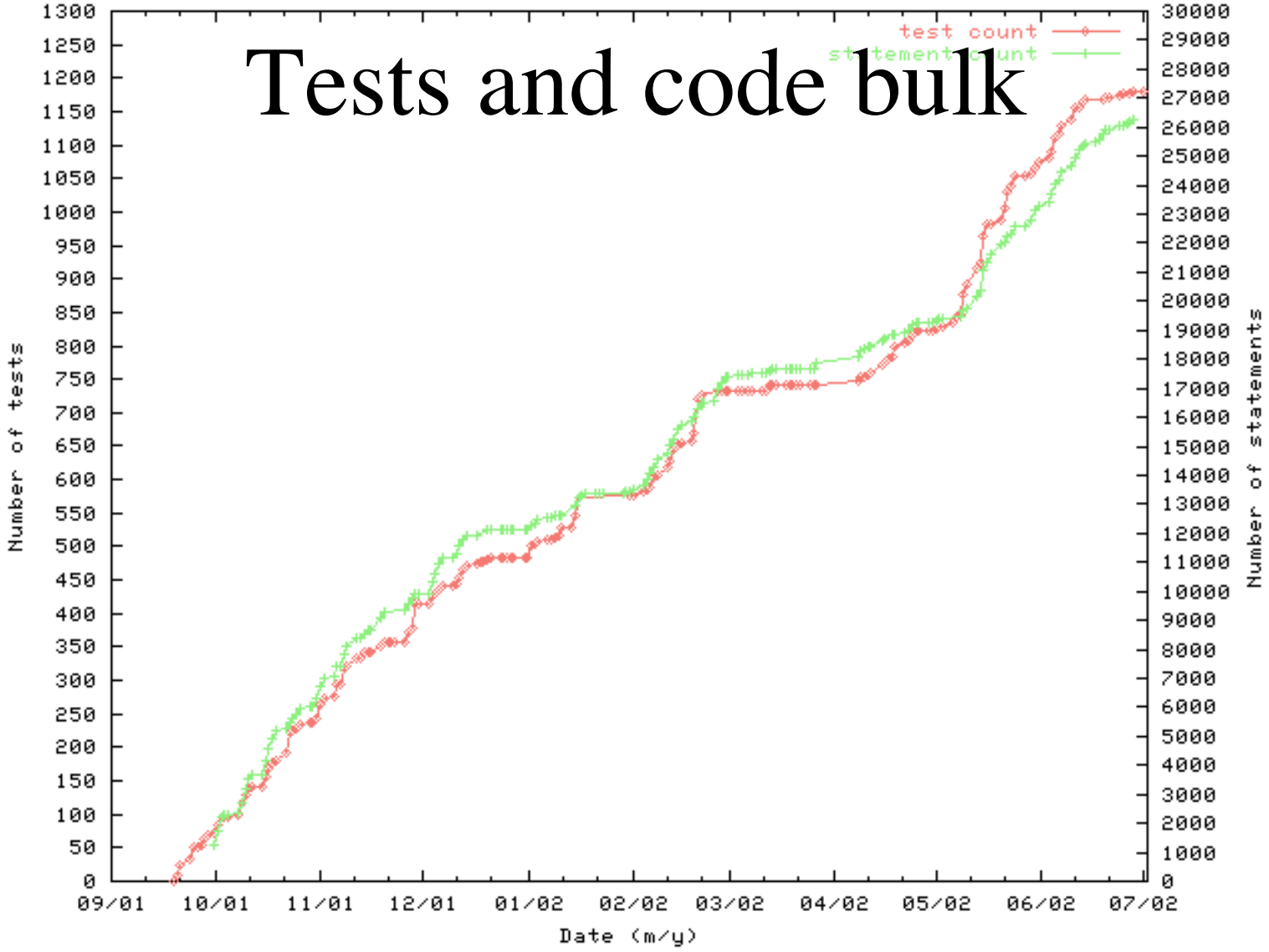
Pragmatics

- Test-parallel and going too deep, too quickly
- “program to an interface” pays off big
  - Example: XMLDataModel, DataSource, SocketDataSource
- Concurrency is hard in tests, too
- Poorly written tests may run standalone
- Tests must be independent of each other
- Tests should leave the system as they find it
- Tests that do too much are easy to think of, hard to write
- Using the file system directly for test resources is bad

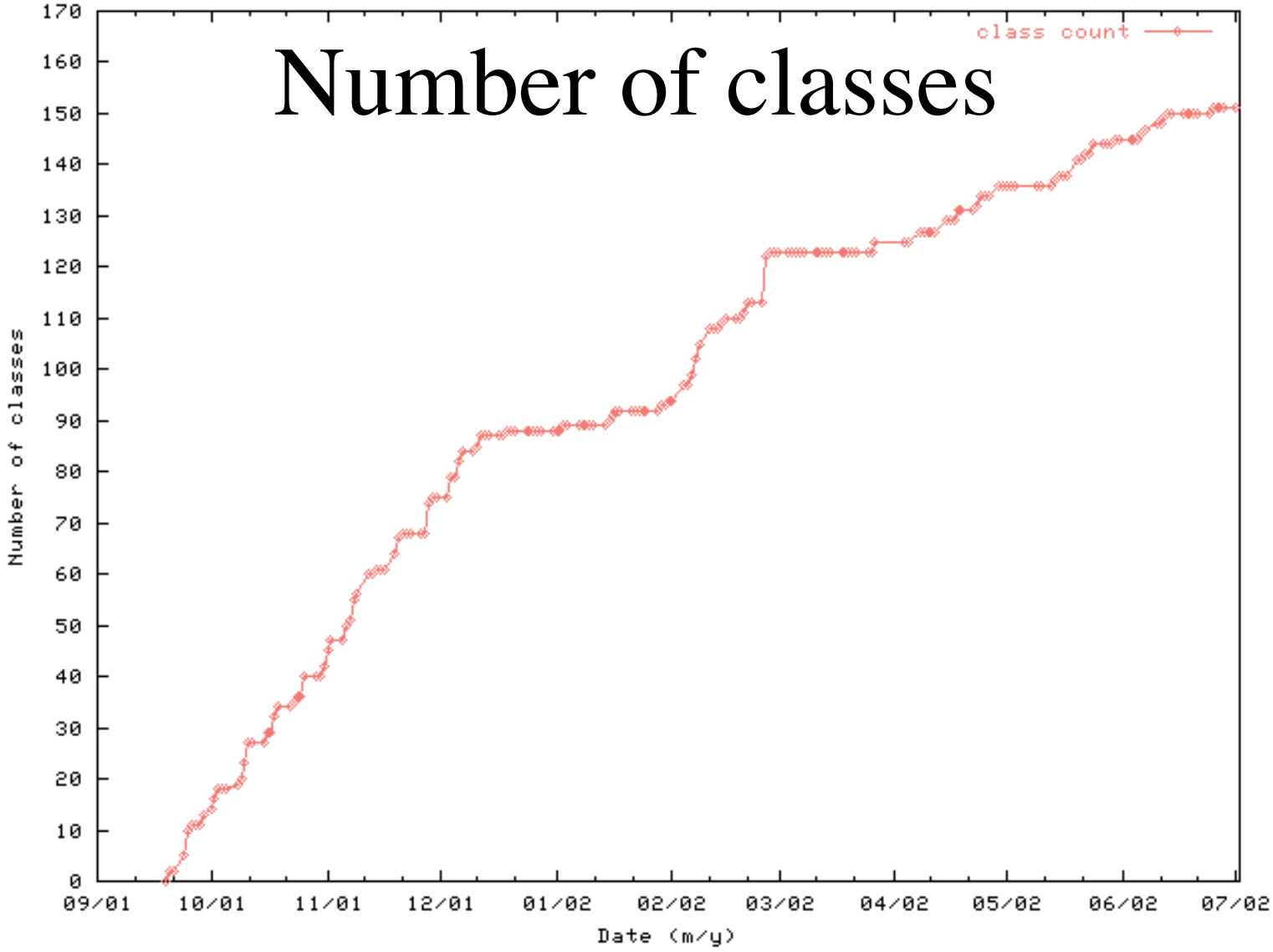
Statistics from a project

- Client for an automotive roll tester
  - 9 months work
  - approximately 2 FTE developers (6 people)
- Source bulk
  - 144 classes
  - 22,600 statements
  - 1055 test methods
  - 2.8 assertions/method

Tests and code bulk



Number of classes



Meaning of data

- Test-parallel development is real, tests grow in parallel with source tree
- Class count plateaus indicate periods when design satisfies requirements
- Improved GUI integration testing towards end of project caused increase in tests