



Developing the SAXESS  
electronic marketplace

---

# About myself

- 1994 KTH-D alumni
- Consultant at Enea Data AB until September of 2000
  - Windows developer (Visual C++)
  - Java developer (Swing, Servlets, EJB)
  - Project manager
- At OM AB since then
  - Subsystem responsible (Java, J2EE)
  - Project manager
  - Process engineer



---

# Agenda

Background

Overview

Development process

Lessons learned



---

# SAXESS - Background

## First Generation:

- Development started 1987
- In production 1989 - 1999
- Trading engine implemented in COBOL on a TANDEM platform
- Could handle up to 10-15 transactions/second (1999)

## Second Generation:

- Development started 1995
- In production since 1999
- Trading engine implemented in ANSI C on a Posix-compliant UNIX platform
- ~ 2000 transactions/second

Currently in production at Stockholm and Copenhagen exchanges





---

# Key Strengths

- **Stability**

- Handle single component failures without service interruption
- Restart after multi-component failure with a minimum of delay

- **Flexibility**

- Possibility to make changes to trading rules and to modify the system configuration without re-programming

- **Functionality**

- Support all basic forms of equity trading, as well as derivatives, mixed-currency trading, combination and contingency order trading

- **Capacity**

- Ability to handle a large number of incoming transactions per second sustained during long periods in time. (depending on hardware performance)



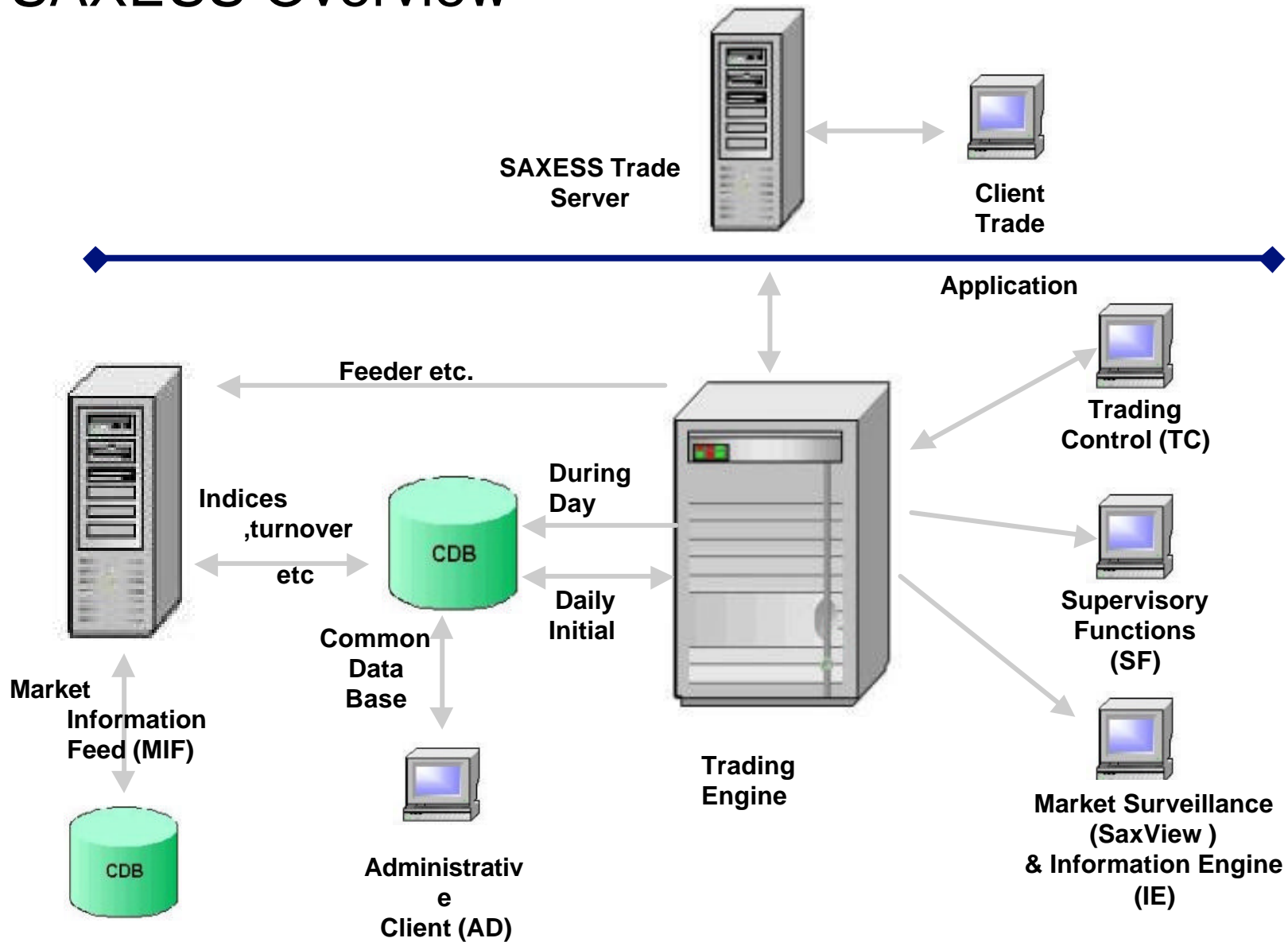
---

# Designed for Flexibility and Performance

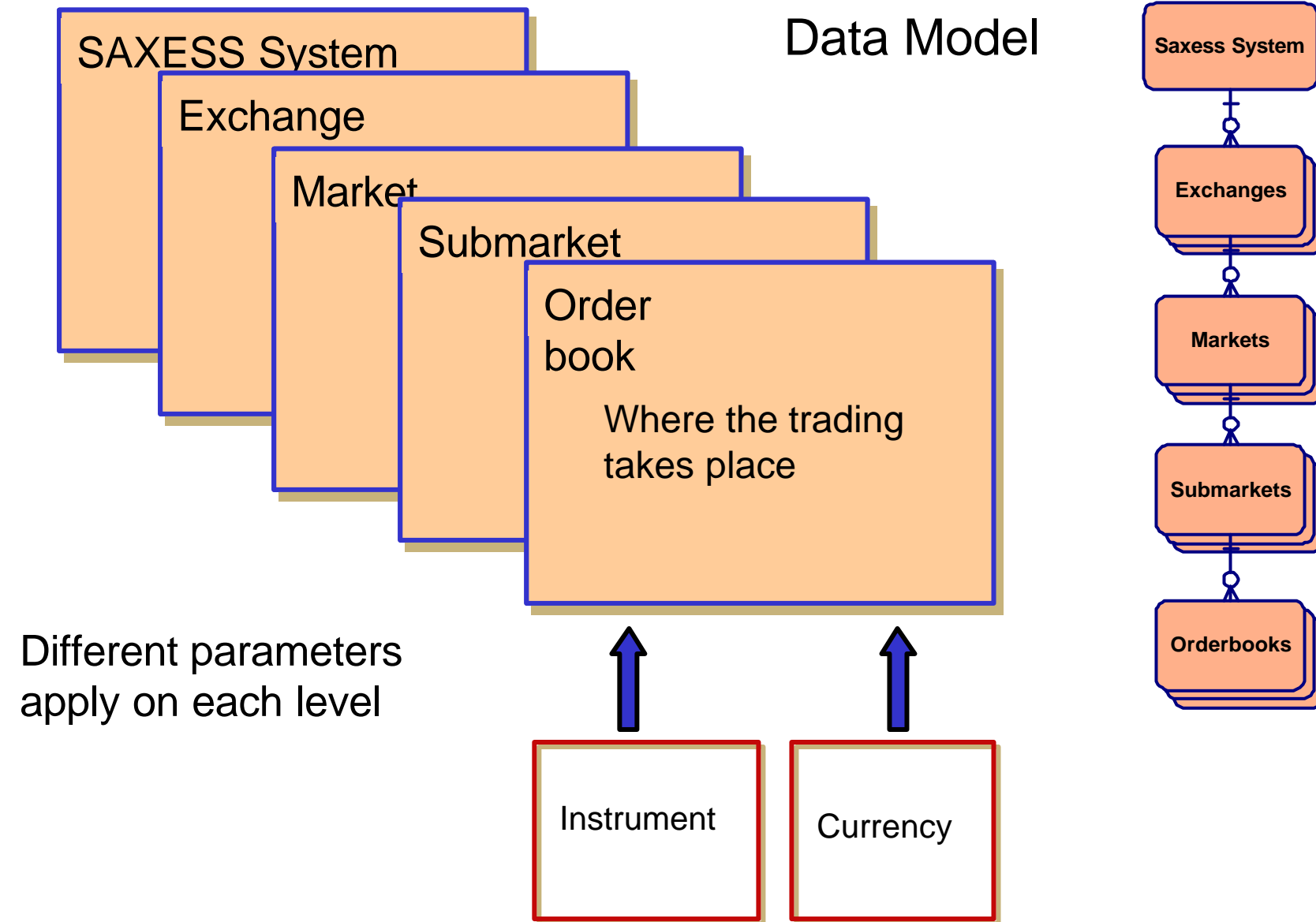
- Portable
- Well-defined interfaces
  - Database (SQL)
  - Communication (TCP/IP)
  - Operating System (Posix)
- Highly configurable
  - Flexible / adaptable data model
  - Data-driven logic
- Performance part of the architecture
  - “Make the common case fast”
  - Few disk accesses (caching)



# SAXESS Overview



# Market Structure



---

# Developing the SAXESS product

## Processes

- OPM - generic project process, customer/legal focus
- SAXESS is implementing internal processes inspired by RUP best practices
  - Develop iteratively
  - Manage requirements
  - Use component architectures
  - Model visually
  - Continuously verify quality
  - Manage change



---

# Project types at SAXESS

- **Customer projects** handles bids, requirements, acceptance testing, deployment, support. There is one customer project per customer and project version.
- **Development projects** handles the development of a new product version.
- Each project is controlled by a steering committee.
- Escalated issues are handled by the SAXESS steering board.



---

# Requirements management

The sales organization generates Bid Projects

- Based on RFO (Requests For Offer)
- 6-12 weeks feasibility study
- Produce FDs (Functional Description)
- Customers receives a DLA (Delivery and License Agreement)

Existing customers generates

- CRs (Change Request)
- RFOs

CCB (Change Control Board) coordinates changes and additions to the product

Resource Allocation Board coordinates projects

The Master Time Schedule contains all projects and major deadlines



---

# Requirement documents

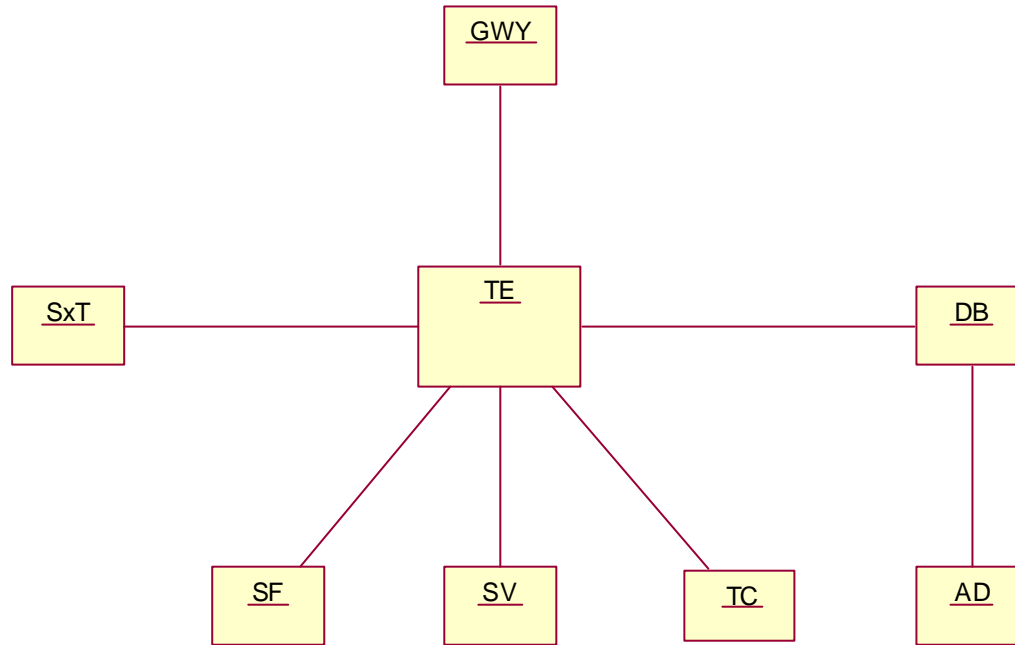
## Functional Description

- Part of the customer contract, contains a background, business case, use case and impact on external protocols
- Basis for implementation planning
- Describes what the system should do on a single function level from the customer perspective

## Functional Specification

- Internal document, contains overall technical solution and scenarios with subsystem interaction based on use case
- Describes the impact on SAXESS subsystems and internal protocols
- Basis for design activities





Msg No	Description	Interface
1	<Describe what message and information is communicated and why this is done.>	<What existing, modified or new message type is used.>
2		

---

# Development

## Design

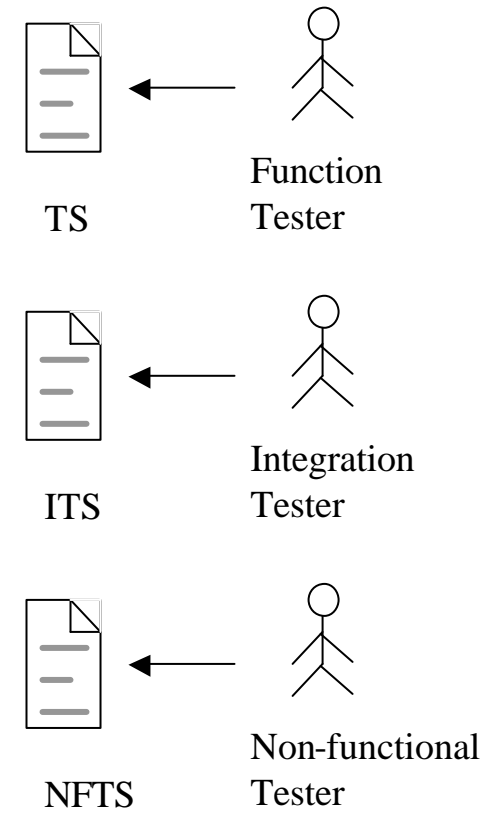
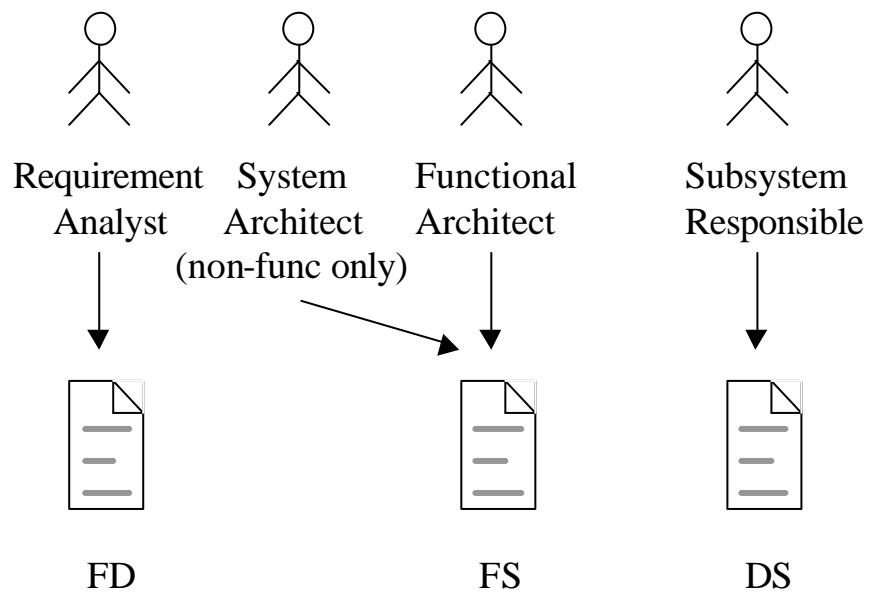
- System specifications provide overall architecture and design information about each subsystem
- Design specifications contain information about the design implications of each FS per subsystem
- Serve as input to test and implementation

## Implementation

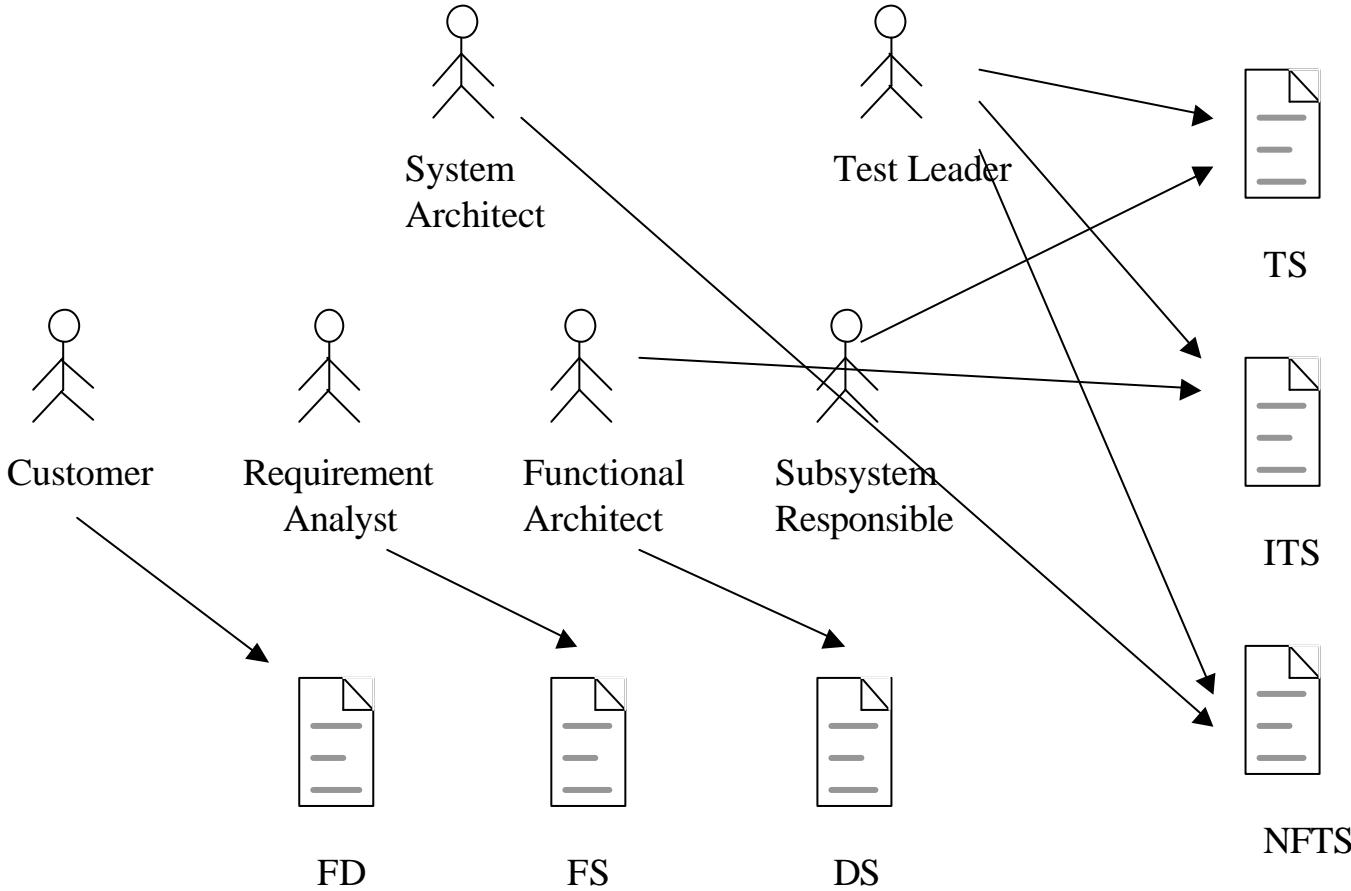
- Heterogeneous environment (C, C++, Java, Delphi)
- Coding standard and code reviews
- CM tools: CVS and VSS



# Document owners



# Document review



---

# The 24-hour cycle

## The trading engine daily build cycle

- New functionality is developed
- Code is committed in CVS
- Build scripts compile the most recent version
- The test engine runs several K test cases and sends report to test engineer
- Test engineer dispatch problem to responsible developer



---

# The test engine

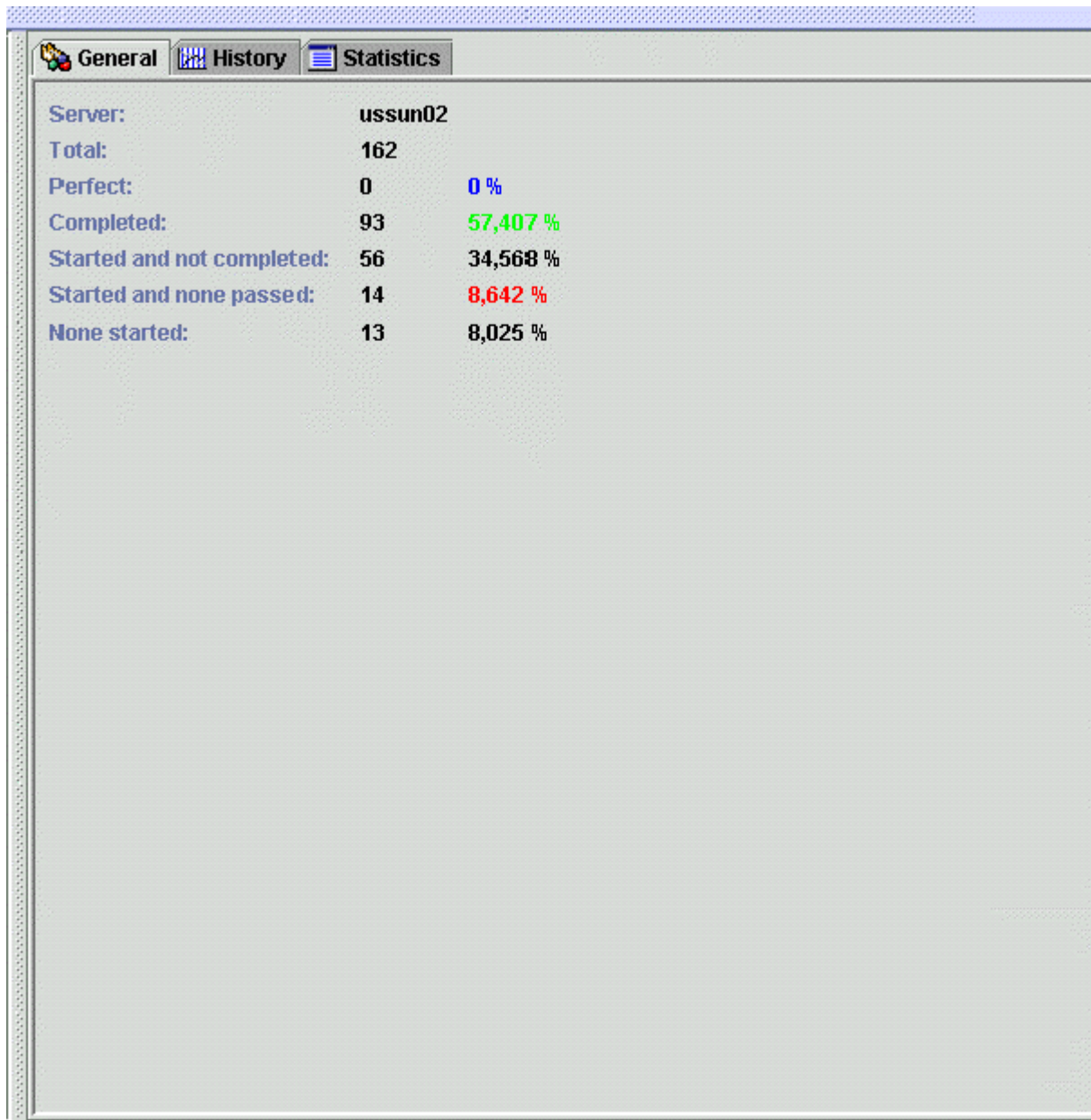
Contains an engine part and a structured library of test cases

- The engine manages the connections with the trading engine through the external protocol XTP
- Each test case is a Java class
- The engine can run any combination of test cases
- Logs from the daily tests are stored for later analysis

Can be used for stress and performance testing

- Simulates production situations by running any number of agents acting as brokers

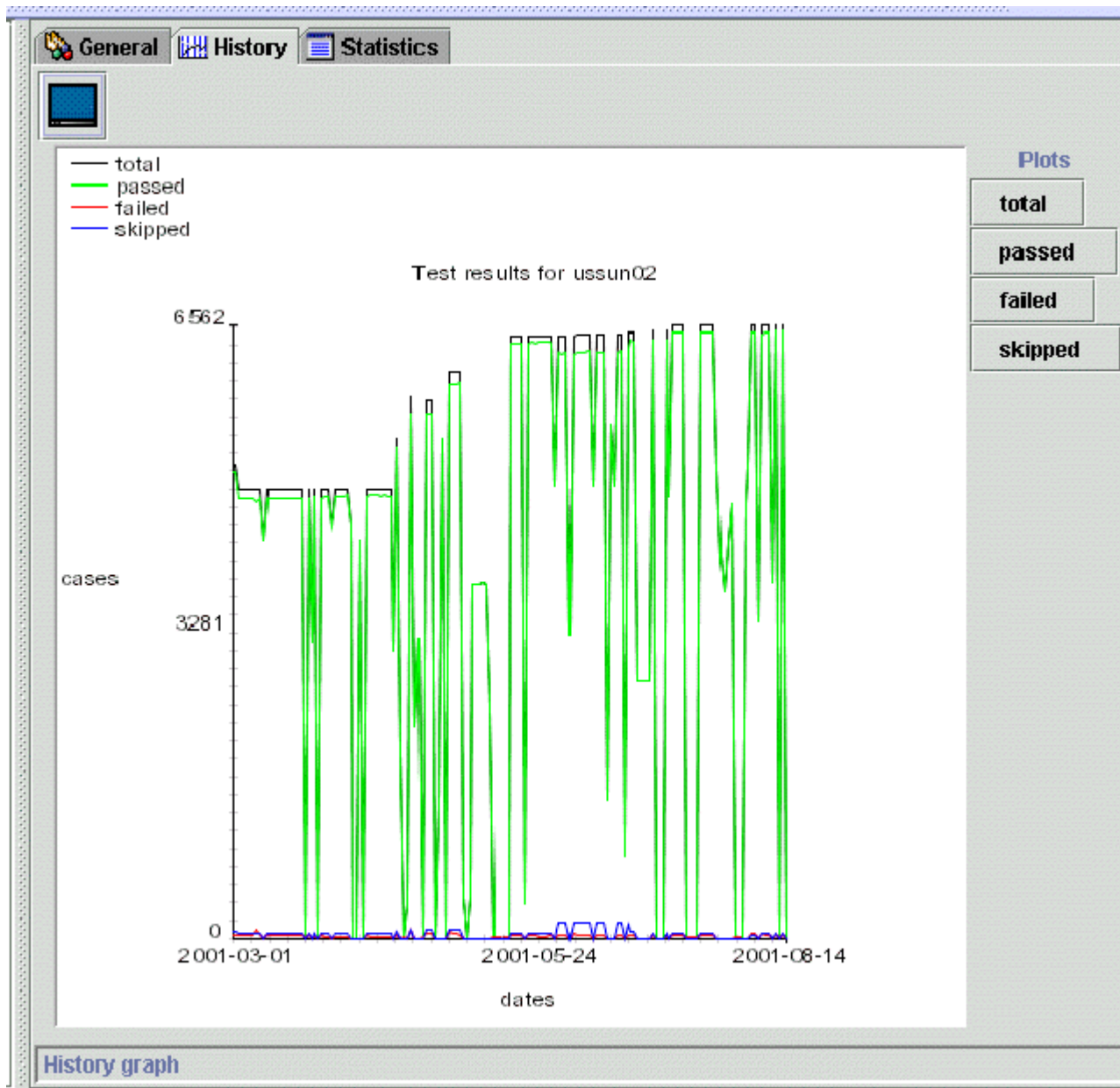




The screenshot shows a software window with three tabs: 'General', 'History', and 'Statistics'. The 'Statistics' tab is active, displaying the following data:

Category	Count	Percentage
Server:	ussun02	
Total:	162	
Perfect:	0	0 %
Completed:	93	57,407 %
Started and not completed:	56	34,568 %
Started and none passed:	14	8,642 %
None started:	13	8,025 %





---

# The one-week cycle

Project management plans the development in increments of one week

- New functionality is developed
- A CVS-tag is made
- The tagged version of the code is built and deployed on the internal test platform
- Integration and regression testing produce problem reports
- The subsystem responsible dispatches the reports
- The developer fixes their assigned problems



---

# Lessons learned at SAXESS

## Requirements specification

- Unambiguous
- Complete

## Resource allocation

- Multi-project environment
- Single project allocation
- Key resources become bottlenecks

