

Putting the Engineering in Software Engineering

■ —————
Nada daVeiga
Parasoft

What is this all about?

- We need to continually build more software to transfer intelligence into computers
- We expect software to change in response to business requirements
- Systems are getting more complicated and harder to control
- However, we still build software as we did many years ago... we have not yet industrialized the process

How is building software different than other manufacturing processes?

- Software is the transfer of human intelligence
- Intelligence is a critical part of the process

What do we need?

- An infrastructure which supports intelligence in this process
 - Relieves humans from repetitive and mundane tasks which can be automated
 - Allows creativity
- Since humans make mistakes, the infrastructure must provide support to correct and eliminate these mistakes

Where does this lead?

- Better software production methods
- Higher productivity
- Better software quality

The Problem with Quality Initiatives

- The current industry focus is on increasing quality
- Managers implementing quality initiatives strive for perfection... but they don't consider the impact on productivity
- Quality initiatives that negatively impact productivity will eventually decay

Productivity > Quality

- Properly implemented quality tasks can:
 - Reduce the amount of work
 - Reduce boredom
 - Allow the team to focus on the creative tasks that truly require human intelligence

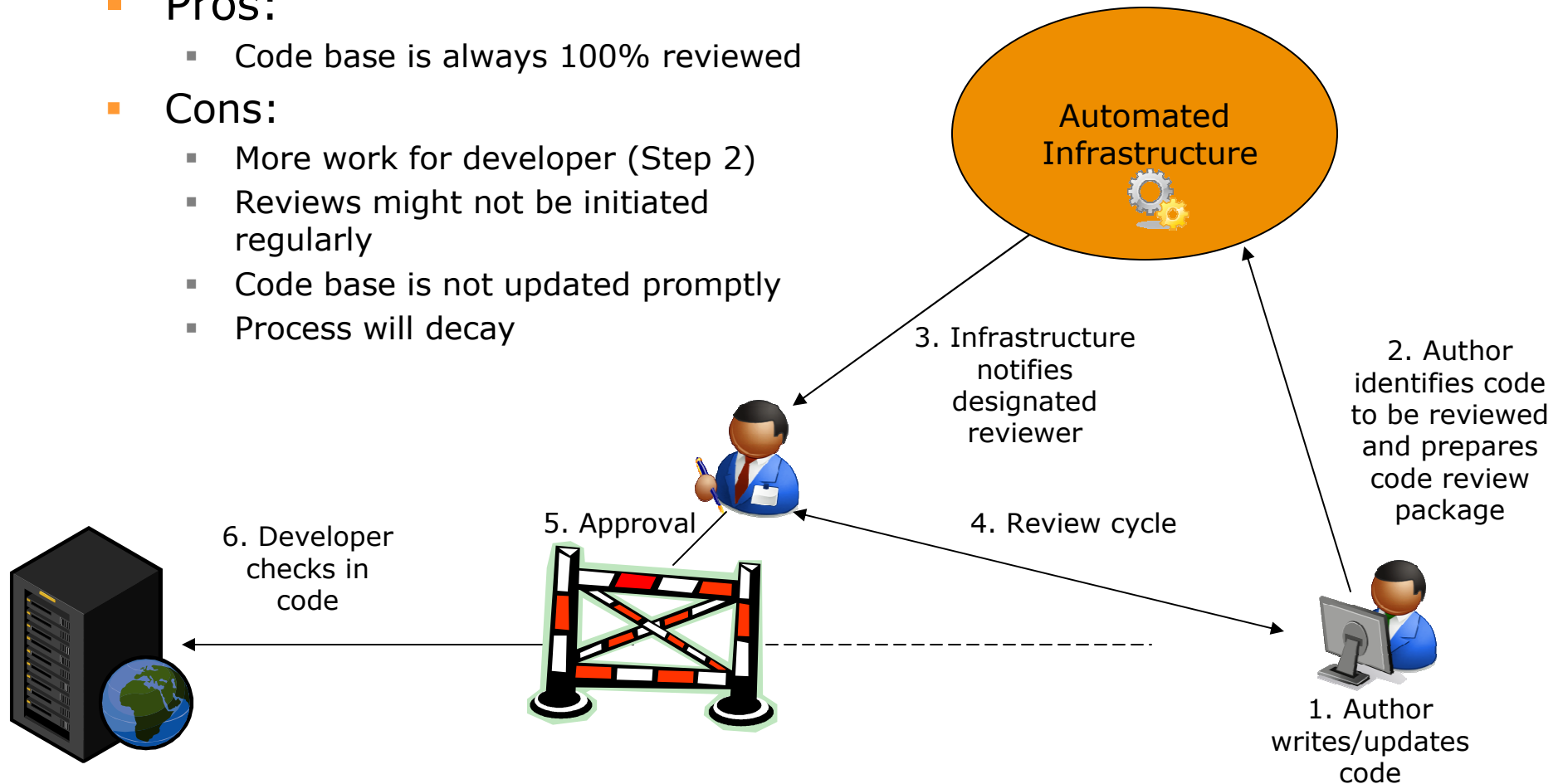
- The result:
 - A sustainable process that delivers greater productivity and significantly fewer software defects

Policy, Workflow, and Infrastructure are Fundamental

- Real quality improvement requires a realistic governance policy and a reactive workflow
 - Quality tasks must save time rather than disrupt workflows/schedules
 - Quality tasks must become a natural part of the workflow
 - Team members should not have to initiate quality tasks— they should just react to problems identified
- Incremental improvement
- The key: an automated infrastructure
 - Drives policies and sustainable workflow
 - Initiates quality tasks
 - Optimizes tasks
 - Checks and measures the entire process

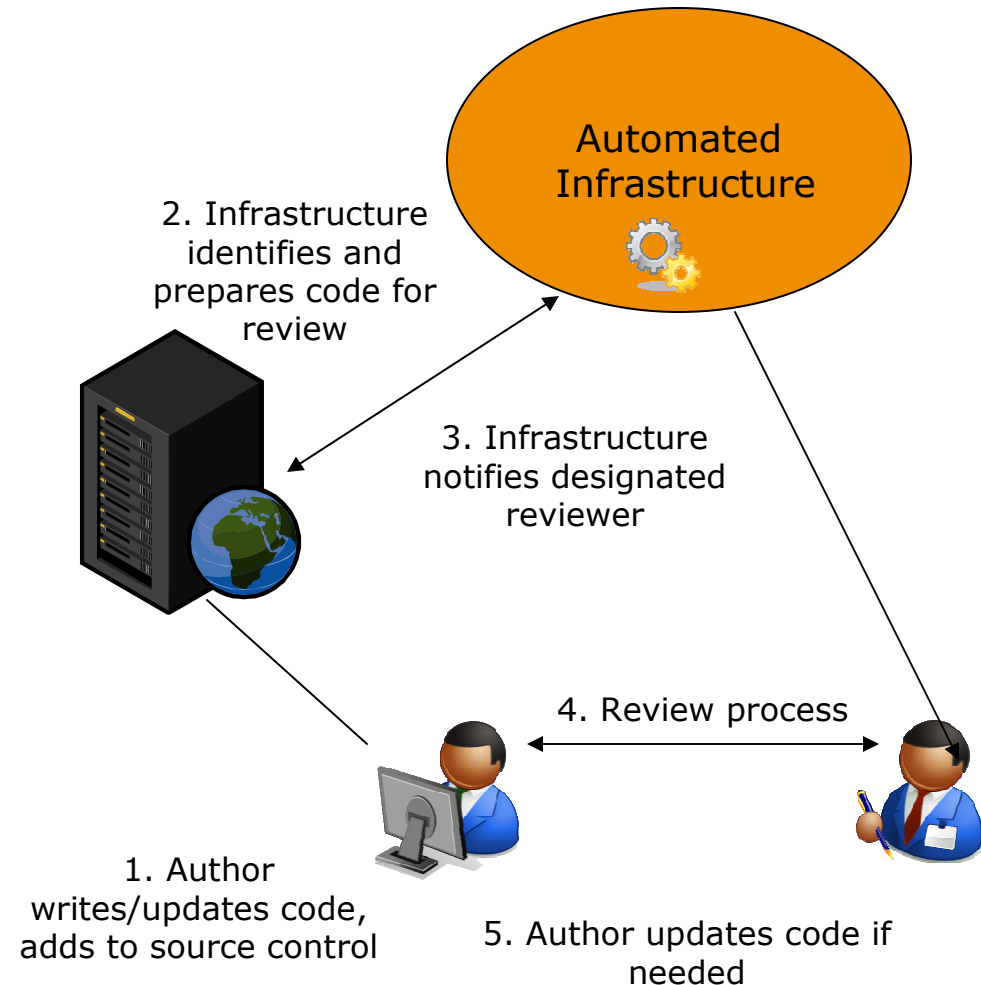
Example: Pre-Commit Code Review

- Pros:
 - Code base is always 100% reviewed
- Cons:
 - More work for developer (Step 2)
 - Reviews might not be initiated regularly
 - Code base is not updated promptly
 - Process will decay



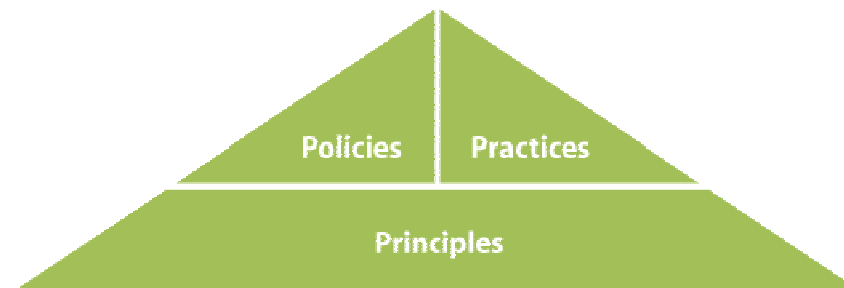
Example: Post-Commit Code Review

- Pros:
 - No added tasks for developer– preparation is initiated and managed automatically (Step 2)
 - Code is prepared for review daily
- Cons:
 - 100% of code in code base might not be reviewed at every moment... but it will all be reviewed within days



How is this done?

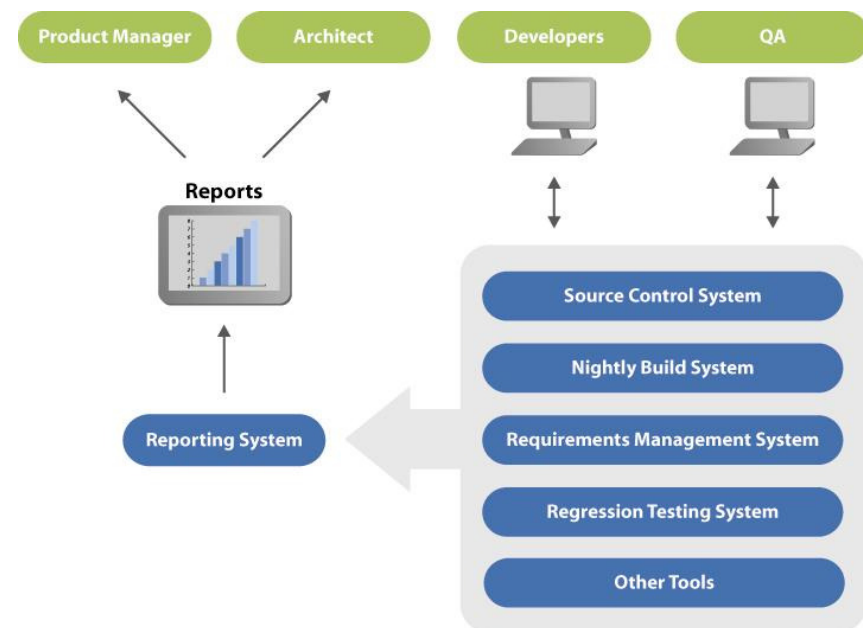
- **Principles:** Basic rules for structuring and managing software projects through defect prevention
- **Practices:** *Functional* embodiments of the principles
- **Policies:** *Managerial* embodiments of the principles



Principle 1: Establishment of Infrastructure

"Build a strong foundation through integration of people and technology"

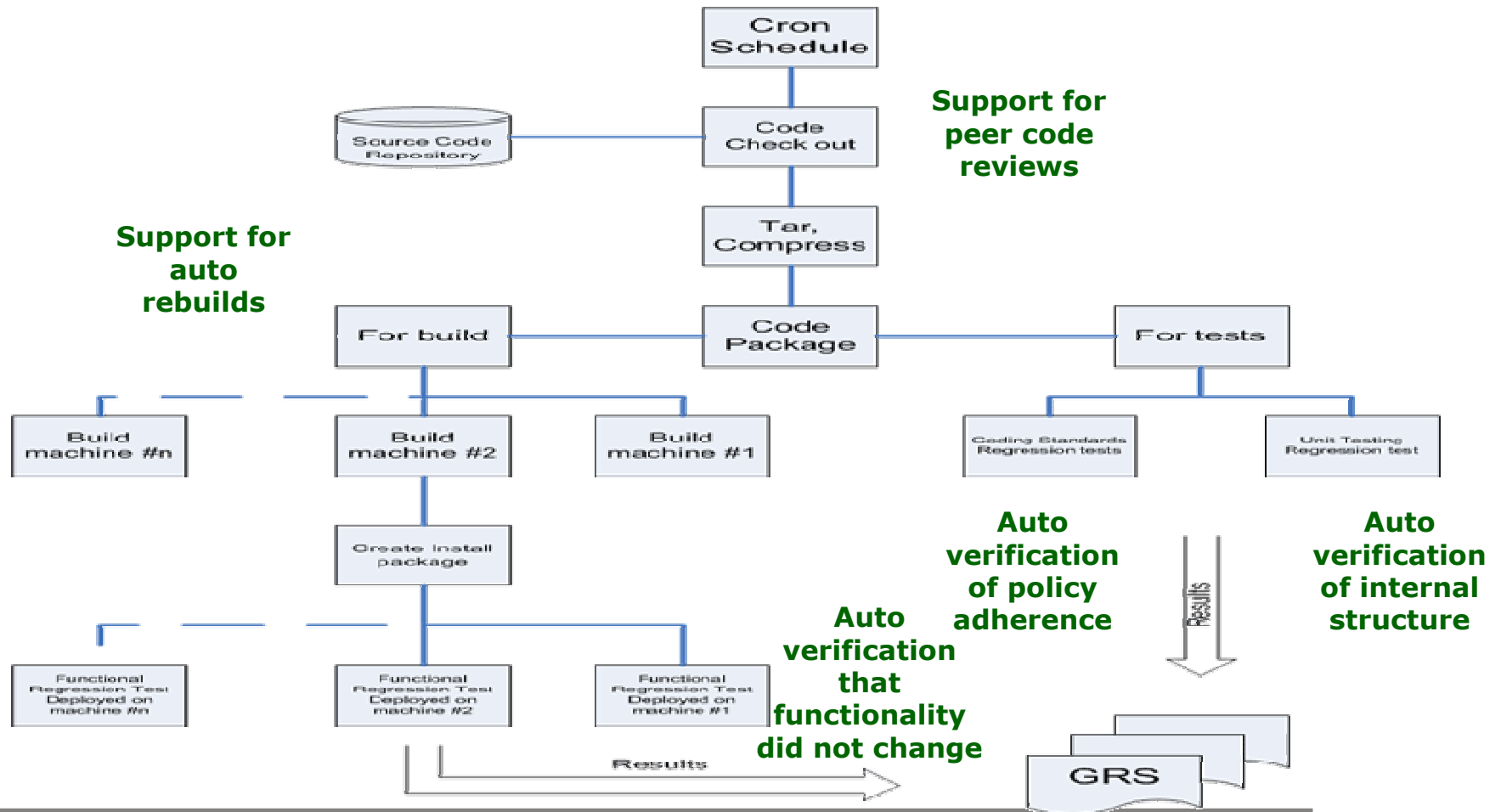
- People
- Configurable technologies
- Integration of the people and the technology



Principle 5: Automation

- *There is no industrialization without automation*
- System complexity makes automation a necessity
- Automation...
 - Improves job satisfaction and productivity
 - Improves product quality
 - Facilitates monitoring
 - Helps to implement and verify best practices and organizational standards
 - Facilitates control of the software processes by collecting measurement data

Infrastructure Supporting Intelligence



Developer Visibility

The screenshot displays the Parasoft JTest IDE interface. On the left, the 'Code Review' window shows a list of review items with dates and reviewer names. Below it, the 'Code Review Issues' window shows a table with columns for 'Source' and 'Revision'. The main editor window shows the source code for 'SQLInjection.java' with a SQL query: `String sQuery = "SELECT * FROM users WHERE name='" + sUserName + "' AND password='" + sPassword + "'";`. The bottom right pane shows a tree view of static analysis tasks, including 'Fix Static Analysis Violations', 'Possible Bugs', 'Security', 'Unused Code', 'Review Assertion Failures', 'Unverified Assertion Failures', 'Severity 2', 'Review Exceptions', and 'Unverified Runtime Exceptions'. The 'Security' task is expanded to show a specific issue: '[2] Protect against SQL injection (BD.SECURITY.TDSQL-1)'.

Source	Revision
struts-config.xml	1.1

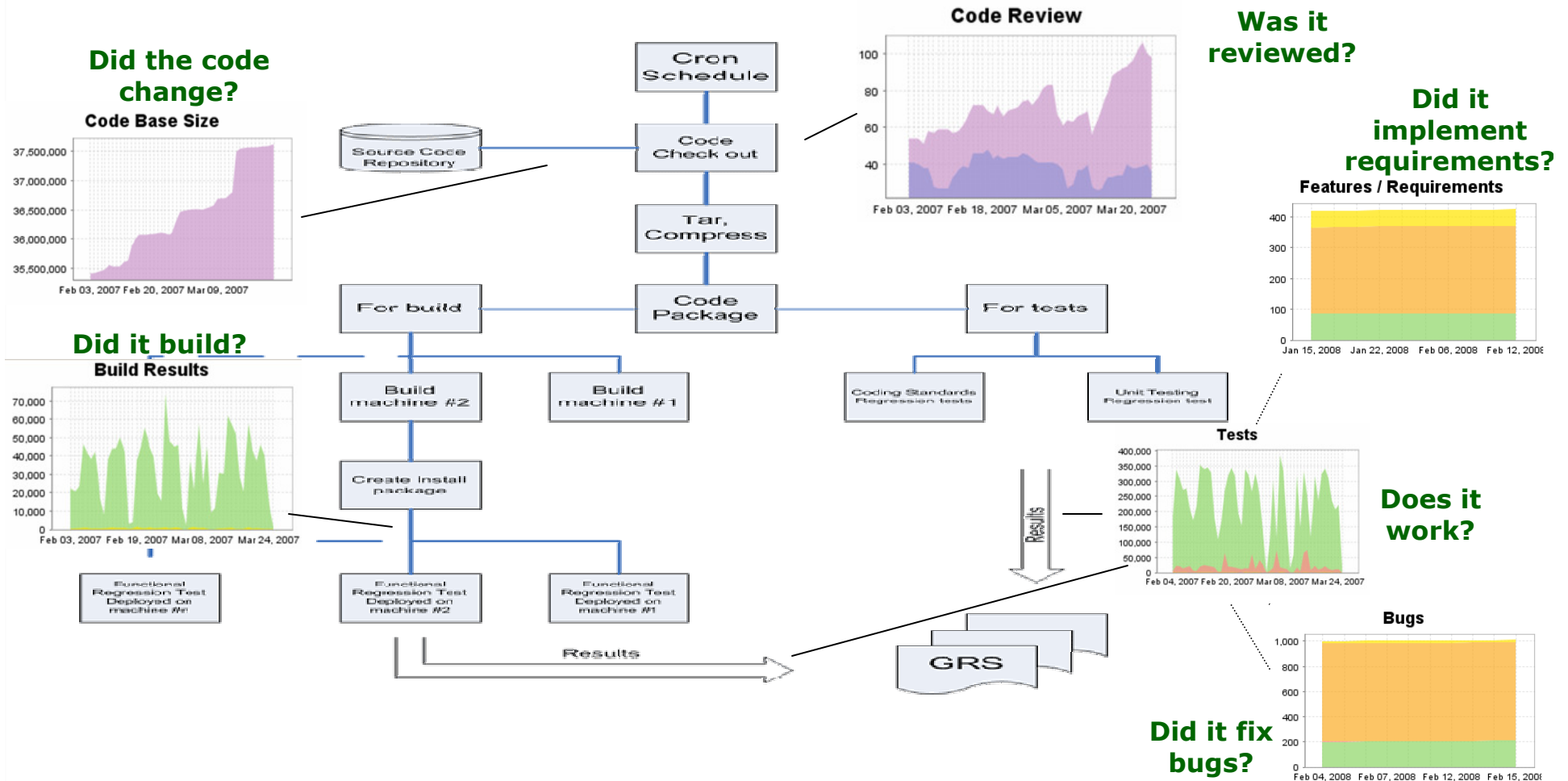
- [7] Jtest Example
 - [7] cynthia
 - [4] Fix Static Analysis Violations
 - [1] Possible Bugs
 - [1] Severity 1
 - [1] Ensure "switch" statements do not contain typos (PB.TLS-1)
 - [2] Security
 - [2] Severity 1
 - [2] Protect against SQL injection (BD.SECURITY.TDSQL-1)
 - [1] Unused Code
 - [2] Review Assertion Failures
 - [2] Unverified Assertion Failures
 - [2] Severity 2
 - [2] junit.framework.AssertionFailedError
 - [2] examples.eval
 - [2] Simple.java
 - [2] [Line 38] junit.framework.AssertionFailedError: expected: <1858> but was: <142>
 - [2] [Line 64] junit.framework.AssertionFailedError: expected: <0> but was: <-10>
 - [1] Review Exceptions
 - [1] Unverified Runtime Exceptions

Principle 4: Measurement and Tracking of Project Status

“Understand the past and present to make decisions about the future”

- Identify problems promptly
- Assess product quality and deployment readiness
- Requires automated reporting system

GRS Visibility



Principle 2: Application of General Best Practices

“Learn from others’ mistakes”

- Best practices from industry experts
- Prevent common errors

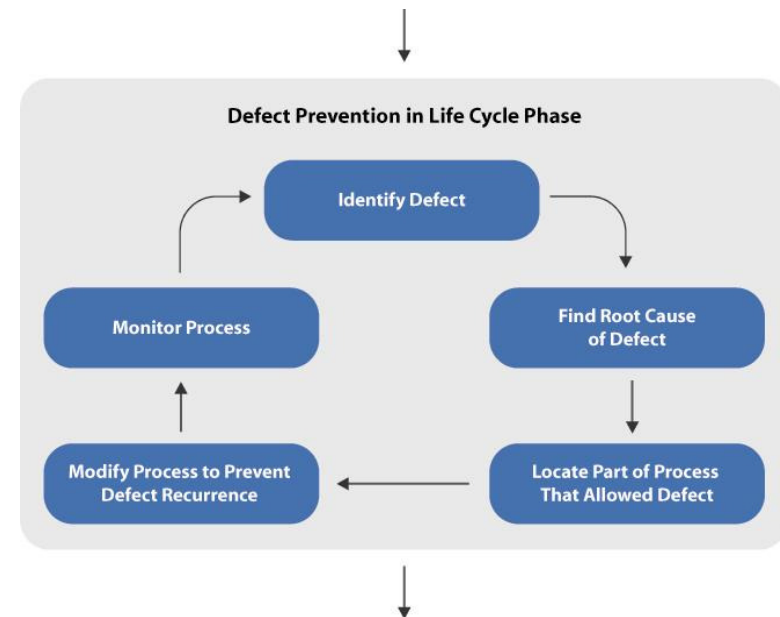
Principle 3: Customization of Best Practices

"Learn from your own mistakes"

- Each time a defect is discovered, a new customized practice is defined and integrated into the process
- Application should be automated and seamless
- Adherence should be monitored

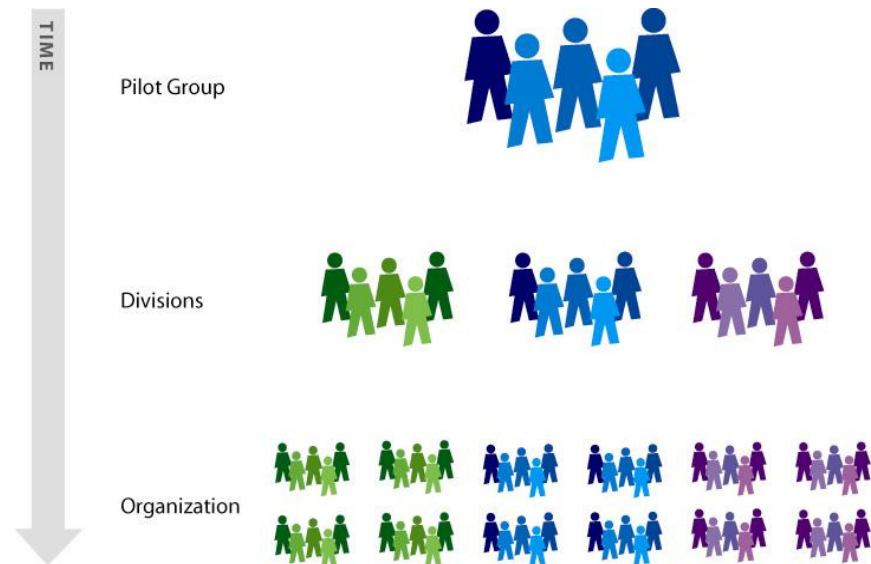
Feedback Loop

1. Identify a defect.
2. Find the root cause of the defect.
3. Locate the point in the production line that caused the defect.
4. Implement preventative practices to ensure that similar defects do not reoccur.
5. Monitor the process to verify effectiveness of the preventive practices.



Principle 6: Incremental Implementation of ADP's Practices and Policies

- Group-by-group
 - Pilot group > expansion
- Practice-by-practice
 - Severity levels
 - Cutoff dates



Summary

Measure and Track



36,000,000 lines of code
18,000,000 tests
9,000,000 comments
9,000,000 source



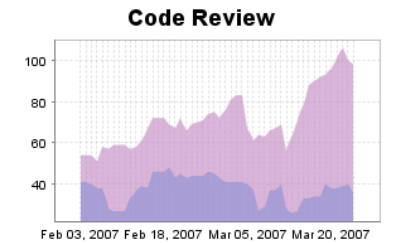
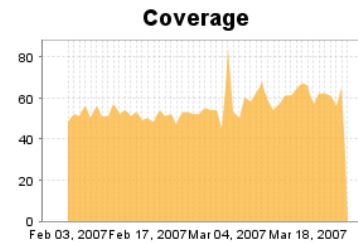
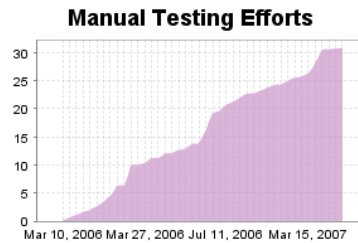
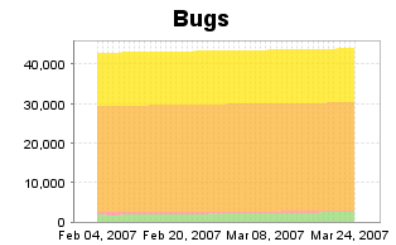
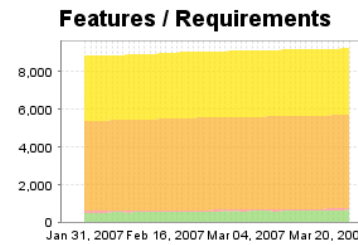
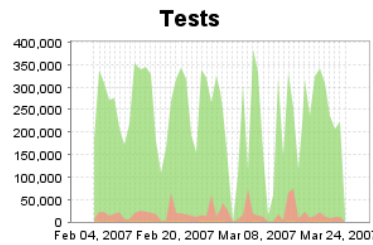
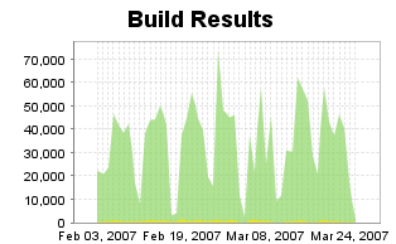
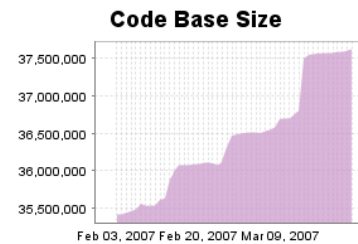
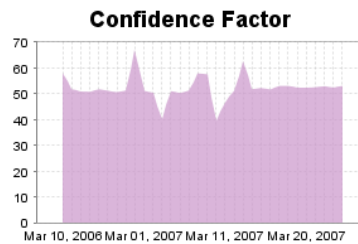
Process Improvement



Productivity Improvement



Quality as
side effect

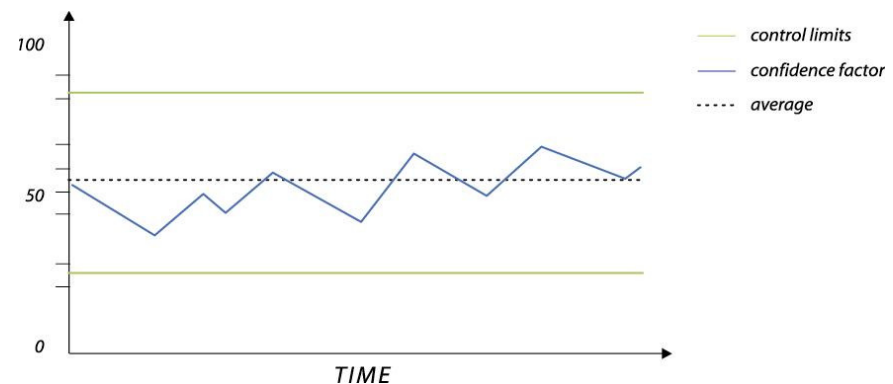


Process Control

- Software processes should be measured and tracked throughout the project lifetime should be treated as statistical processes
- Use data to evaluate process trends; as the project progresses, the trends in its processes should stabilize
- The goal: a predefined level of stabilization and capability
 - A **stable process** is predictable; its variation is under control (for example, when representative variables are plotted on a control chart, they fall between the upper control limit and the lower control limit, which are based on quality controls such as Six Sigma)
 - For a process to be considered **capable**, it must be stable and the average of its plotted variables must fall within the specification limits, which vary for each process.

Process Control Example - Confidence Factor

- Used to evaluate software quality and help in making deployment decisions
- An aggregate of measures such feature/requirement implementation index, test pass rate, code check in stability, and others
- CF should stay within small range near the top of the scale
 - Code is not being broken through feature additions, the test cases are succeeding, etc.
 - The application can be released
- The graphed process is stable, but not yet capable



Raising the Confidence Factor

- Study all indicators to determine the weakest points
- Example: If weakest point is violations of coding standards, the code quality should be increased
 - This would increase the CF, but further work might be required
 - Other problems might exist, such as too many code modifications and uncompleted features with their consequent failing tests
 - Continue the cycle of identifying and fixing problems until the CF reaches the appropriate level – this is necessary to attain product stability and capability and arrive at an informed release decision

Parasoft Nucleus

