Institute for Software Research
UNIVERSITY OF CALIFORNIA, IRVINE

Understanding and Improving Software Productivity

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425 USA
www.ics.uci.edu/~wscacchi

16 February 2005

Introduction

- What affects software productivity?
  - Software productivity has been one of the most studied aspects of software engineering
  - *Goal:* review sample of *empirical studies* of software productivity for large-scale software systems from the 1970's through the early 2000's.
- How do we improve software productivity?
  - Looking back (history)
  - Looking forward (future)

Understanding and improving software productivity: *Historic view*

Preview of findings

- Most software productivity studies are inadequate and misleading.
- How and what you measure determines how much productivity you see.
- Small-scale programming productivity has more than *an order of magnitude* variation across individuals and languages
- We find contradictory findings and repeated shortcomings in productivity measurement and data analysis, among the few nuggets of improved understanding.

Basic software productivity dilemma

- *What to measure?*
- Productivity is usually expressed as a *ratio*
  - Outputs/Inputs
  - This assumes we know what the units of output and input are
  - This assumes that both are continuous and linear (like "real numbers", not like "weather temperatures")

Software productivity dilemma

- We seek to understand what affects and how to improve software productivity
  - Measurement is a quest for certainty and control
  - What role does measurement take in helping to improve software productivity?
- Measurement depends on instrumentation, so the relationship must be clear.
- Instrumentation choices lead to *trade-offs*.

Measurement-instrumentation trade-offs

- Who/what should perform measurement?
- What types of measurements to use?
- How to perform the measurements?
- How to present results to minimize distortion?
- Most software productivity studies assume *ratio* measurement data is preferred.
  - However, *nominal*, *ordinal*, or *interval* measures may be very useful.
- Thus, what types of measures are most appropriate for understanding software productivity?

Why measure software productivity?

- Increase software production productivity or quality
- Develop more valuable products for lower costs
- Rationalize higher capital-to-staff investments
- Streamline or downsize software production operations
- Identify production bottlenecks or underutilized resources
- But trade-offs exist among these!

Who should measure software productivity?

- Programmer self-report
- Project or team manager
- Outside analysts or observers
- Automated performance monitors
- Trade-offs exist among these

What to measure?

- Software products
- Software production processes and structures
- Software production setting

Software products

- Delivered source statements, function points, and reused/external components
- Software development analyses
- Documents and artifacts
- Application-domain knowledge
- Acquired software development skills with product or product-line

Software production processes

- *Requirements analysis*: frequency and distribution of requirements changes, and other *volatility* measures.
- *Specification*: number and interconnection of computational objects, attributes, relations, and operations in target system, and their volatility.
- *Architectural design*: design complexity; the volatility of the architecture's configuration, version space, and design team composition; ratio of new to reused architectural components.
- *Unit design*: design effort; number of potential design defects detected and removed before coding.
- *Coding*: effort to code designed modules; ratio of inconsistencies found between module design and implementation by coders.
- *Testing*: ratio of effort allocated to spent on module, subsystem, or system testing; density of known error types; extent of automated mechanisms employed to generate test case data and evaluate test case results.

Software production setting

- Programming language(s)
- Application type
- Computing platforms
- Disparity between host and target platforms
- Software development environment
- Personnel skill base
- Dependence on outside organizations
- Extent of client or end-user participation
- Frequency and history of mid-project platform upgrades
- Frequency and history of troublesome anomalies and mistakes in prior projects

Findings from software productivity studies

- More than 30 empirical studies of software productivity have been published
  - Aerospace, telecommunications, insurance, banking, IT, and others
  - Company studies, laboratory studies, industry studies, field studies, international studies, and others
- A small sample of studies
  - ITT Advanced Technology Center (1984)
  - USC System Factory (1990)
  - IT and Productivity (1995)

ITT Advanced Technology Center

- Systematic data on programming productivity, quality, and cost was collected from 44 projects in 17 corporate subsidiaries in 9 countries, accounting for 2.3M LOC and 1500 person years of effort.
- *Finding:* product-related and process-related factors account for approximately the same amount (~33%) of productivity variance.
- *Finding:* you can distinguish productivity factors that can be controlled (process-related) from those that cannot (product-related).

ITT productivity factors

Process-related factors (more easily controlled)

- avoid hardware-software co-development
- development computer size (bigger is better)
- Stable requirements and specification
- use of "modern programming practices"
- assign experienced personnel to team

Product-related factors (not easily controlled)

- computing resource constraints (fewer is better)
- program complexity (less is better)
- customer participation (less is better)
- size of program product (smaller is better)

USC System Factory

- Examined the effect of teamwork in developing both formal and informal software specifications.
- *Finding:* observed variation in productivity and specification quality could be best explained in terms of *recurring teamwork structures.*
  - Six teamwork structures (patterns of interaction) were observed across five teams; teams frequently shifted from one structure to another.
- High productivity and high product quality results could be traced back to observable patterns of teamwork.
- Teamwork structures, cohesiveness, and shifting patterns of teamwork are salient productivity variables.
- See S. Bendifallah and W. Scacchi, Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork, *Proc. 11th. Intern. Conf. Software Engineering* , Pittsburgh, PA, IEEE Computer Society, 260-270, May 1989.

Work structures and shifts (data)

Team ID	T1	T2	T3	T4	T5
Size	6	7	7	7	5
Reusable Exemplar	no	yes	yes	yes	yes
PROCESS					
A. Pre-planning task	N->R->I	N->R->I	N->R->I	N->R->I	N->R->I
B. Planning task	N =>	N =>	N =>	N =>	N =>
c.	I	I	I	I -> S	I -> S
d.	I	I+	I	S	S
e.	P (D,I,I)	P (D,I,I)	P (D,I,I)	P (D,S->I,I)	P (D,D,I)
f.	D	D	D	D	D
g.	R	R	R	D	D
h.	D	I	I	D	D
C. Develop preliminary	I -->>	I+	I+	I+ -> S+	I -> S+
(informal) specification	N -->>				
	R -->> I				
D. Develop formal	I+	I+	I+	S+ -->>	S+
(processable) spec.				N -->>	
E. Document	N -->> D	P+	P+	N -->>	P(D,D,I)
write-up				P(D,S->I,D)	
F. Documentation	D -->>	D+	D+	D+	D+
integration	N -->> I				
G. Document review	R	R	R	N -->> R	D
H. Prepare for	D -->>	I+	N -->> I+	N -->> I+	D+
Delivery	N -->> I				

IT and Productivity

- *IT* is defined to include software systems for transaction processing, strategic information systems, and other applications.
- Examines studies drawn from multiple economic sectors in the US economy.
- *Finding:* apparent "productivity paradox" in the development and use of IT is due to:
  - *Mismeasurement* of inputs and outputs.
  - *Lags* due to adaptation and learning curve effects.
  - *Redistribution* of gains or profits.
  - *Mismanagement of IT* within industrial organizations.
- Thus, one significant cause for our inability to understand software productivity is found in *mismeasurement*.

Summary: Software Productivity Drivers

- What affects software productivity?
  - Software development environment attributes
  - Software system product attributes
  - Project staff attributes

Software development environment attributes

- Provide substantial (and fast!) computing resource infrastructure
- Use contemporary SE tools and techniques
- Employ development aids that help project coordination
- Use "appropriate" (domain-specific) programming languages
- Employ process-center development environments

Software system product attributes

- Develop small-to-medium complexity systems
- Reuse software that already addresses the problem
- No real-time or distributed software to develop
- Minimal constraints for validation of accuracy, security, and ease of modification
- Stable requirements and specifications
- Short task schedules to avoid slippages

Project staff attributes

- Small, well-organized project teams
- Experienced development staff
- People who collect their own productivity data
- Shifting patterns of teamwork structures

How to improve software productivity (so far)

- Get the best from well-managed people.
- Make development steps more efficient and more effective.
- Simplify, collapse, or eliminate development steps.
- Eliminate rework.
- Build simpler products or product families.
- Reuse proven products, processes, and production settings.

Summary of software productivity measurement challenges

- Understanding software productivity requires a large, complex set of qualitative and quantitative data from multiple sources.
- The number and diversity of variables indicate that software productivity cannot be understood simply as a ratio source code/function points produced per unit of time/budget.
- A more systematic understanding of interrelationships, causality, and systemic consequences is required.
- *We need a more robust theoretical framework, analytical method, and support tools to address these challenges*

Understanding and improving software productivity: *Future view*

A knowledge management approach to software engineering

- Develop setting-specific theories of software production
- Identify and cultivate local software productivity drivers
- Develop knowledge-based systems that model, simulate, re-enact, and redesign software development and usage processes
- Develop, deploy, use, and continuously improve a computer-supported cooperative *organizational learning* environment

Develop setting-specific theories of software production

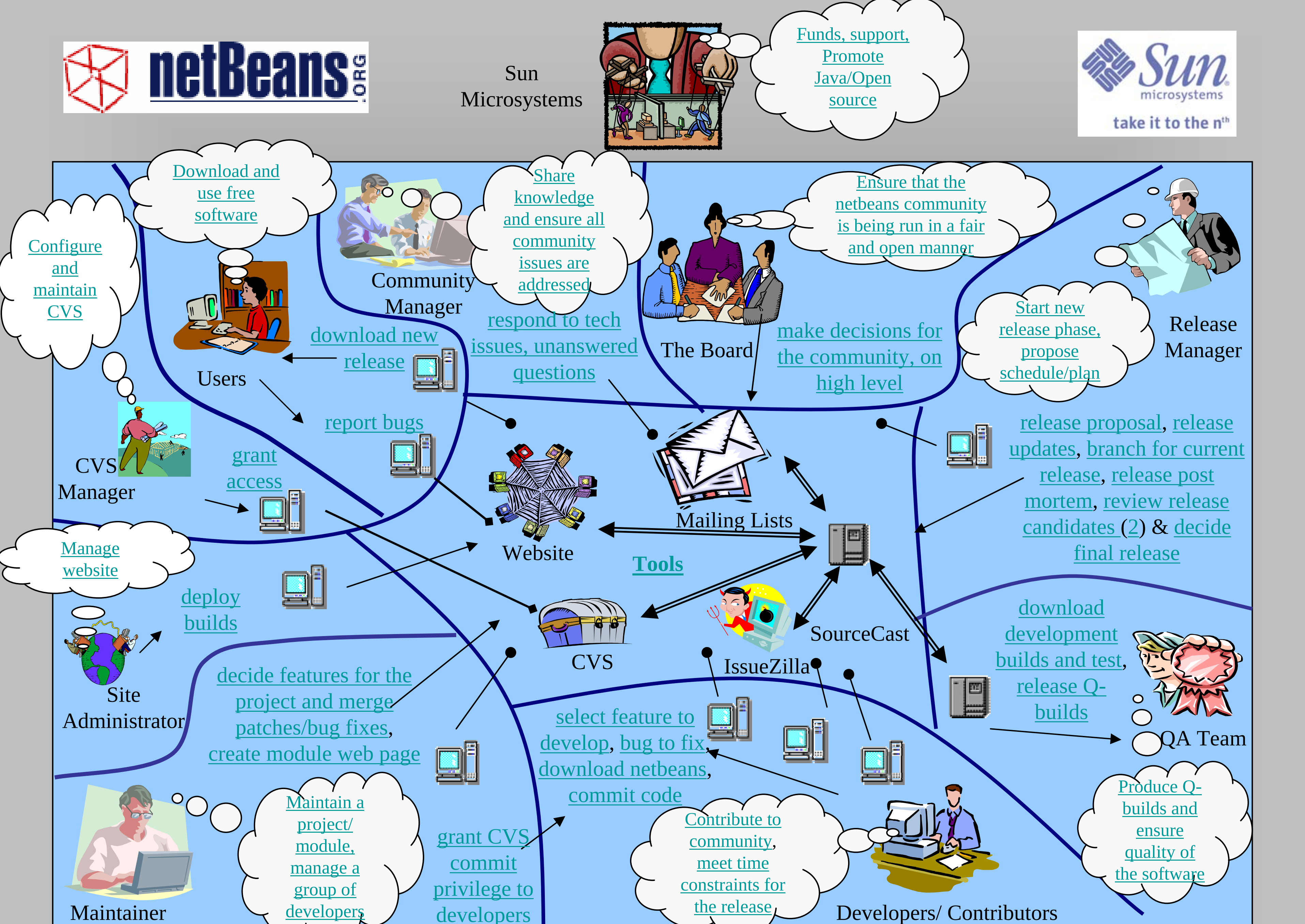
- Conventional measures of software product attributes do little in helping to understand software productivity.
- We lack an articulated *theory of software production*.
- We need to construct *models*, hypotheses, and measures that account for software production in different settings.
- These models and measures should be *tuned* to account for the mutual influence of software products, processes, and setting characteristics specific to a development project.
- We need field study efforts to contribute to this

Identify and cultivate software productivity drivers

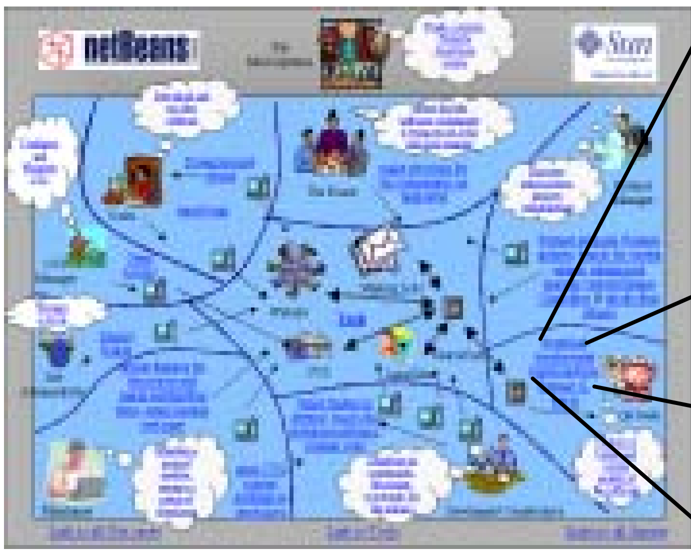
- Why are software developers so productive in the presence of technical and organizational constraints?
- Software developers must realize the potential for productivity improvement.
  - The potential for productivity improvement is not an inherent property of new software development technology.
  - Technological impediments and organizational constraints can nullify this potential.
- Thus, a basic concern must be to identify and cultivate *software productivity drivers*.
  - Examples include workplace incentives and alternative software business models

Model, simulate, re-enact, and redesign software development and usage processes

- New software process modeling, analysis, and simulation technology is becoming available.
- Knowledge-based software process technology supports capture, description, and application of causal and interrelated knowledge about what can affect software development (from field studies).
- Requires an underlying computational model of process states, actions, plans, schedules, expectations, histories, etc. in order to answer dynamic "what-if" questions.



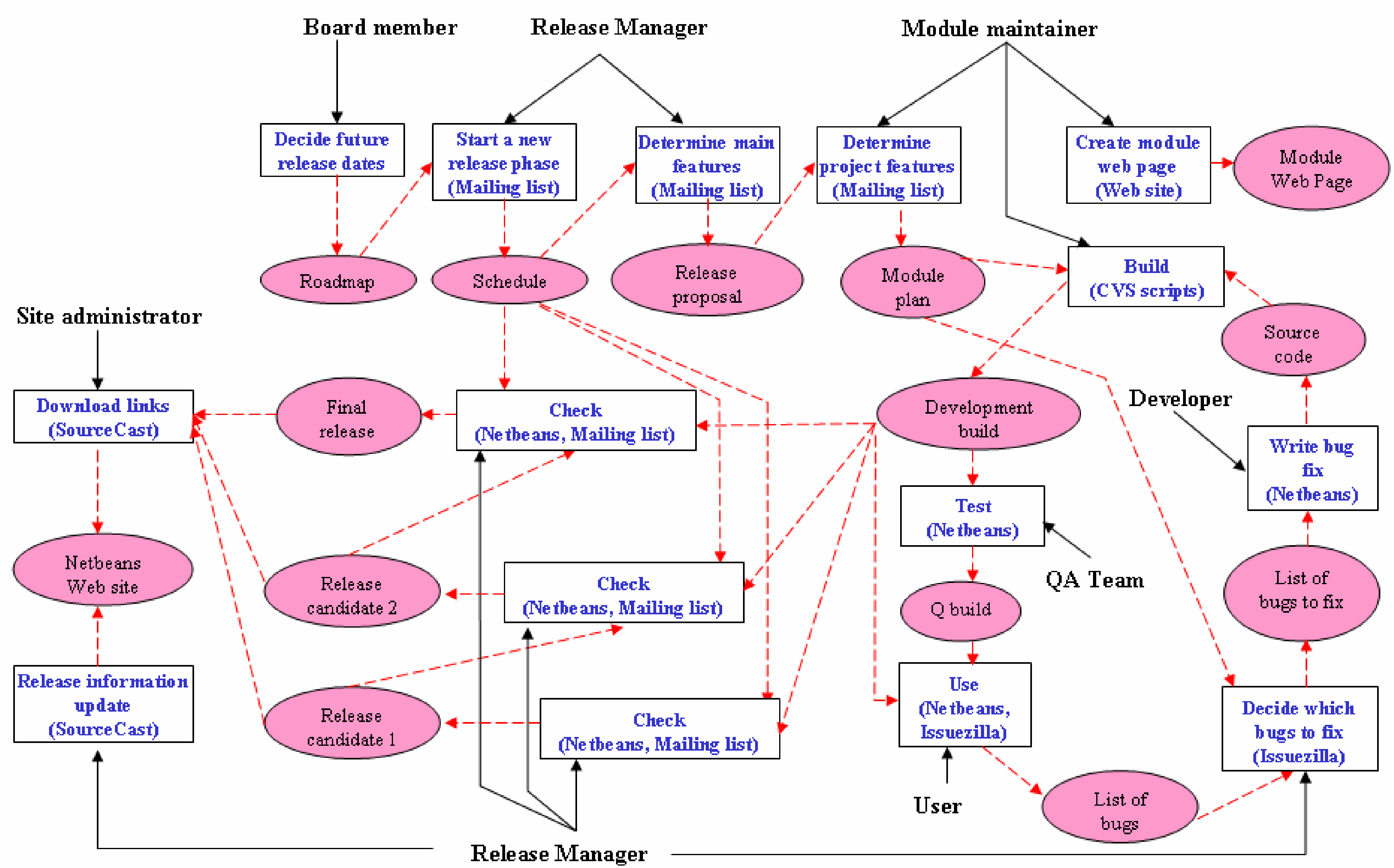
[Link to all Use Cases](#) [Link to Tools](#) [Links to all Agents](#)

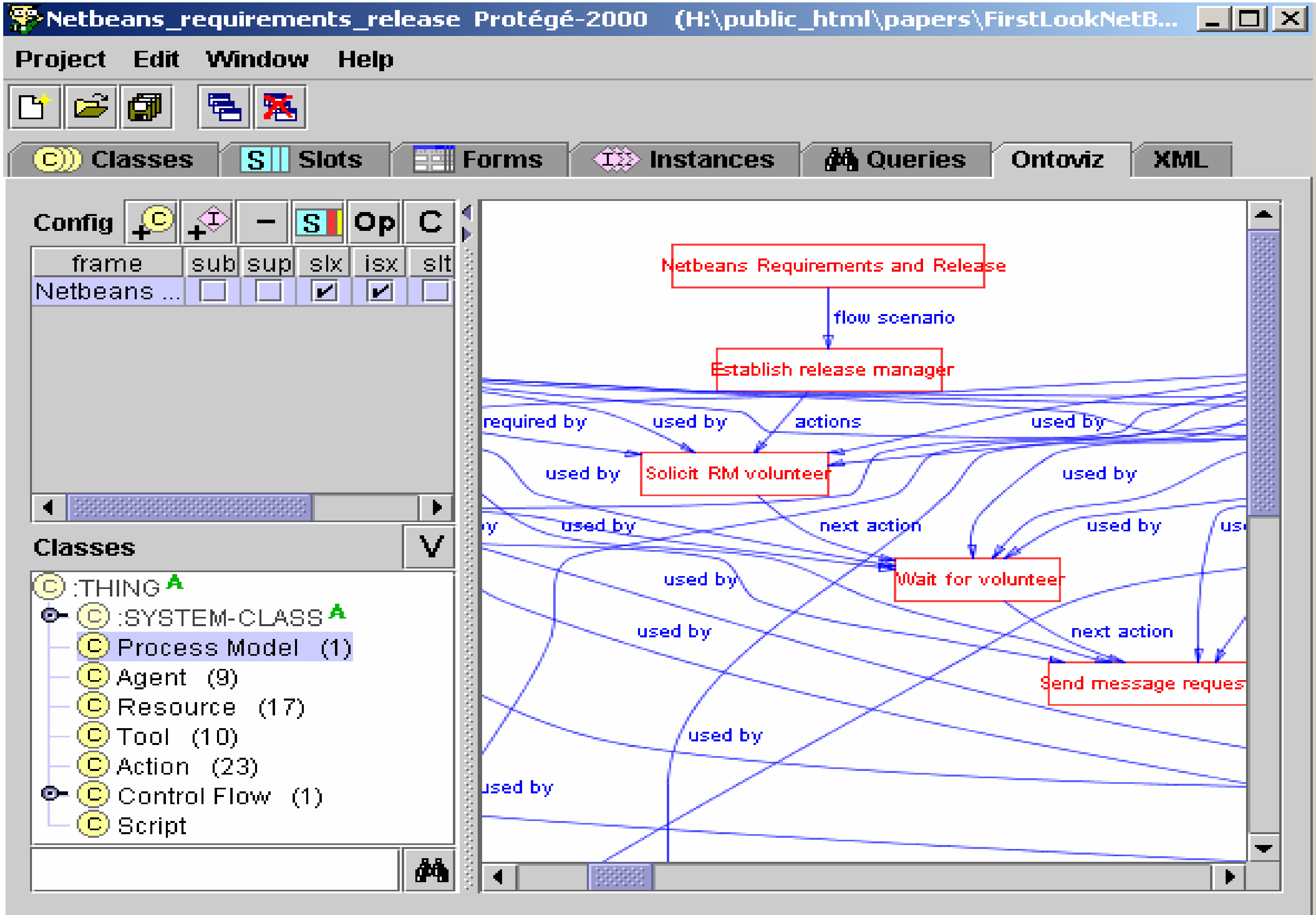


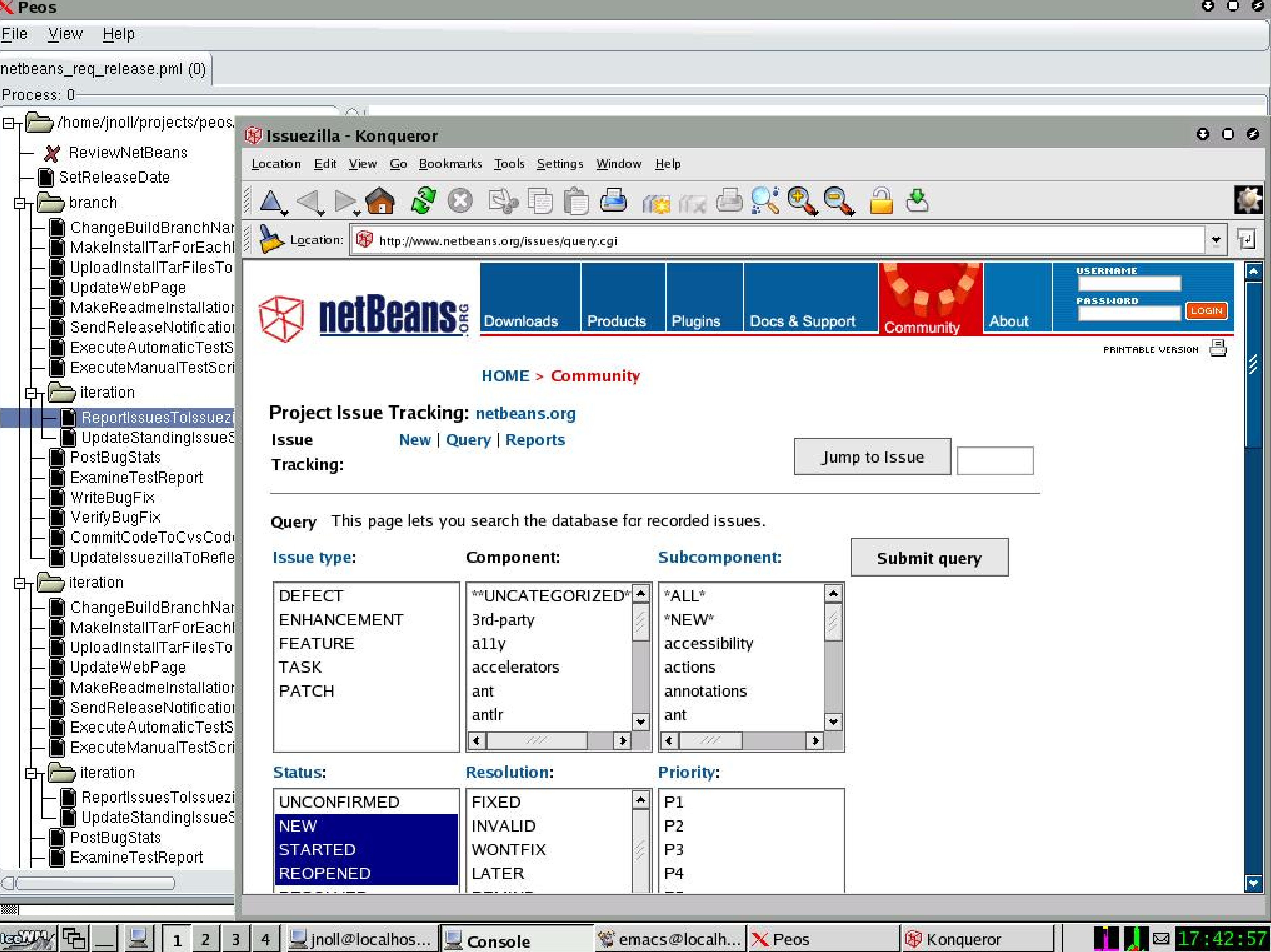
Test Builds

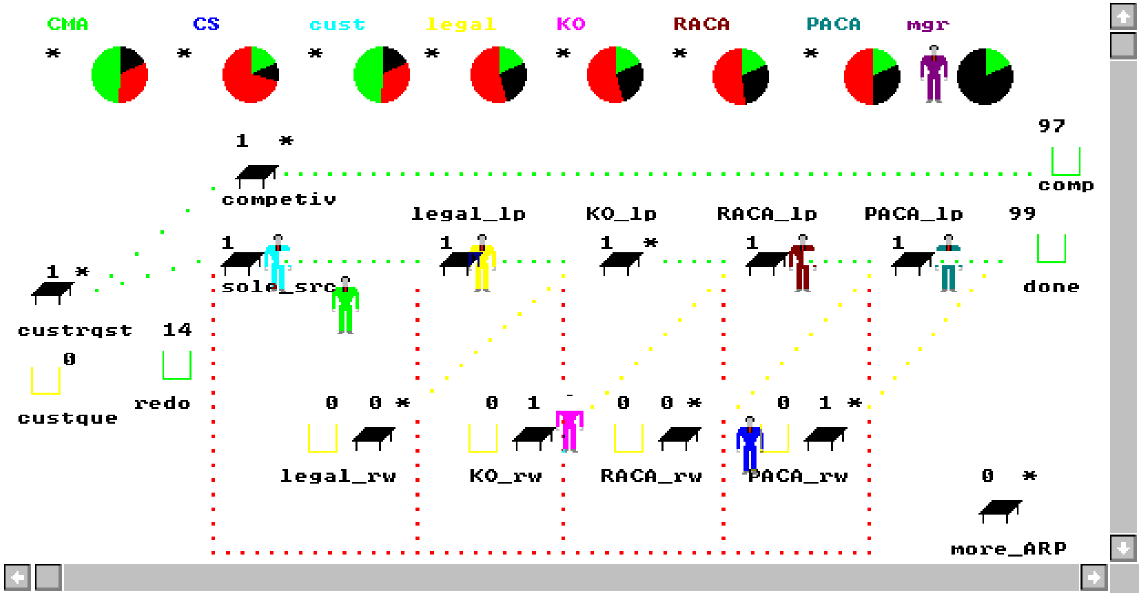
- **The QA team tests the latest nightly builds every Friday**
  - **QA team executes a set of manual tests on the builds as well as some sanity checks**
- **Test results are categorized as**
  - **Bug Types**
- ***User Constraint:***
  - **The tests depend on the manual tests specification**
- ***System Constraint:***
  - **Not all bugs may be identified**

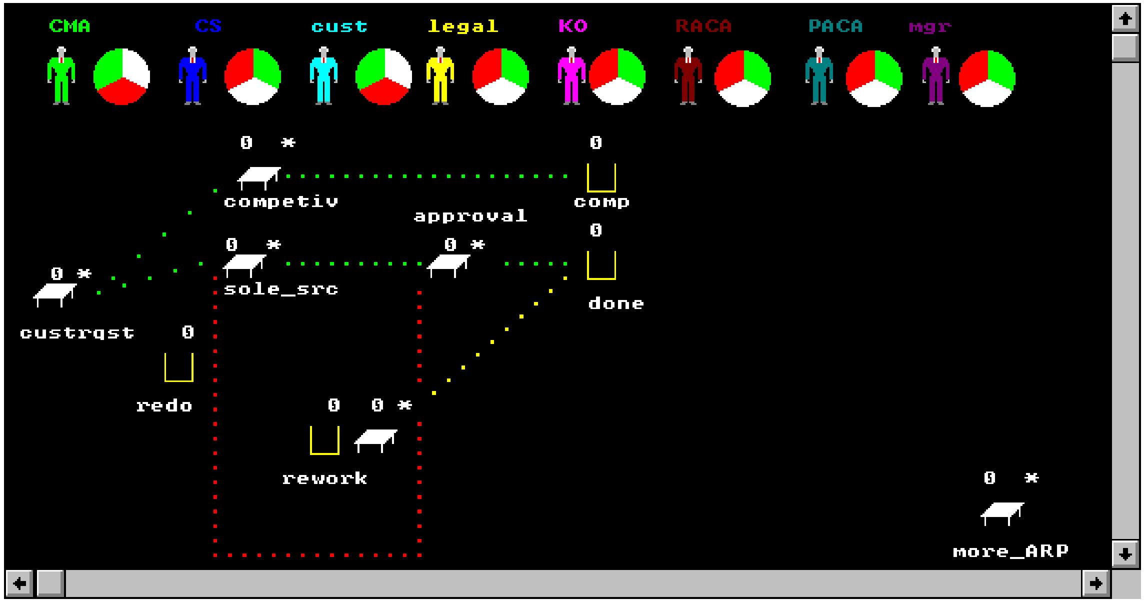
Figure 2. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case.


Board member
Release Manager
Module maintainer
Decide future release dates
Start a new release phase (Mailing list)
Determine main features (Mailing list)
Determine project features (Mailing list)
Create module web page (Web site)
Module Web Page
Roadmap
Schedule
Release proposal
Module plan
Build (CVS scripts)
Source code
Site administrator
Download links (SourceCast)
Final release
Check (Netbeans, Mailing list)
Development build
Developer
Write bug fix (Netbeans)
Test (Netbeans)
QA Team
Netbeans Web site
Release candidate 2
Check (Netbeans, Mailing list)
Q build
List of bugs to fix
Release information update (SourceCast)
Release candidate 1
Check (Netbeans, Mailing list)
Use (Netbeans, Issuezilla)
Decide which bugs to fix (Issuezilla)
List of bugs
User
Release Manager


Netbeans_requirements_release Protégé-2000 (H:\public_html\papers\FirstLookNetB...
Project Edit Window Help
Classes
Slots
Forms
Instances
Queries
Ontoviz
XML
Config
Op
C
frame
sub
sup
slx
isx
slt
Netbeans ...
Classes
:THING
:SYSTEM-CLASS
Process Model (1)
Agent (9)
Resource (17)
Tool (10)
Action (23)
Control Flow (1)
Script
Netbeans Requirements and Release
flow scenario
Establish release manager
required by
used by
actions
Solicit RM volunteer
next action
Wait for volunteer
Send message reques

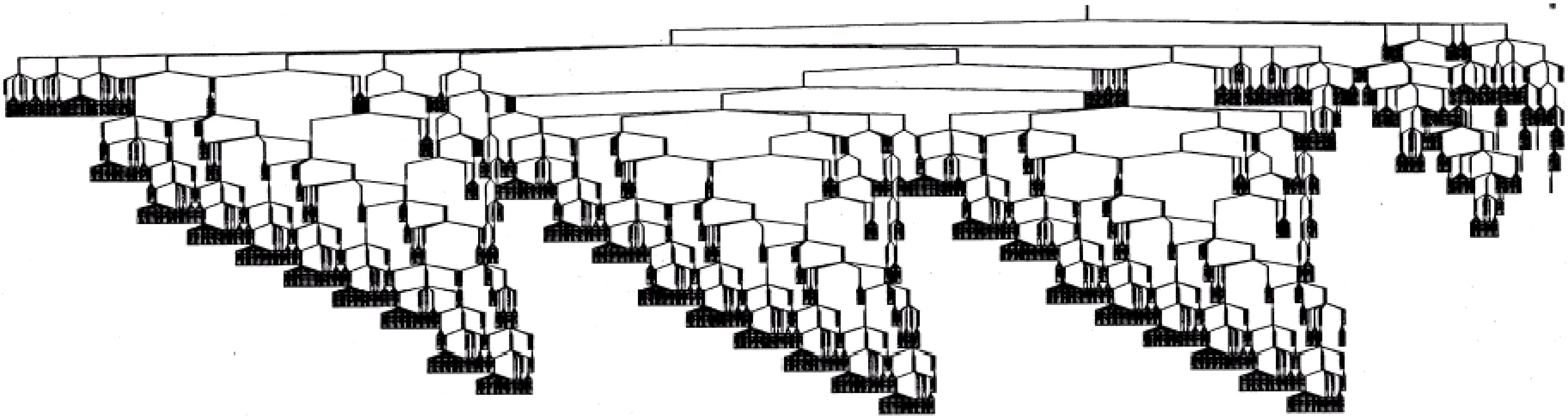

Peos
File View Help
netbeans_req_release.pml (0)
Process: 0
/home/jnoll/projects/peos
ReviewNetBeans
SetReleaseDate
branch
ChangeBuildBranchNar
MakeInstallTarForEachI
UploadInstallTarFilesTo
UpdateWebPage
MakeReadmeInstallatior
SendReleaseNotificatior
ExecuteAutomaticTestS
ExecuteManualTestScri
iteration
ReportIssuesToIssuezi
UpdateStandingIssueS
PostBugStats
ExamineTestReport
WriteBugFix
VerifyBugFix
CommitCodeToCvsCode
UpdateIssuezillaToRefle
iteration
ChangeBuildBranchNar
MakeInstallTarForEachI
UploadInstallTarFilesTo
UpdateWebPage
MakeReadmeInstallatior
SendReleaseNotificatior
ExecuteAutomaticTestS
ExecuteManualTestScri
iteration
ReportIssuesToIssuezi
UpdateStandingIssueS
PostBugStats
ExamineTestReport
Issuezilla - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://www.netbeans.org/issues/query.cgi
netBeans.ORG
Downloads
Products
Plugins
Docs & Support
Community
About
USERNAME
PASSWORD
LOGIN
PRINTABLE VERSION
HOME > Community
Project Issue Tracking: netbeans.org
Issue Tracking:
New | Query | Reports
Jump to Issue
Query This page lets you search the database for recorded issues.
Issue type:
Component:
Subcomponent:
Submit query
DEFECT
ENHANCEMENT
FEATURE
TASK
PATCH
UNCATEGORIZED
3rd-party
a11y
accelerators
ant
antlr
ALL
NEW
accessibility
actions
annotations
ant
Status:
Resolution:
Priority:
UNCONFIRMED
NEW
STARTED
REOPENED
FIXED
INVALID
WONTFIX
LATER
P1
P2
P3
P4
1 2 3 4
jnoll@localhos...
Console
emacs@localh...
Peos
Konqueror
17:42:57


CMA
CS
cust
legal
KO
RACA
PACA
mgr
competiv
comp
97
legal_lp
KO_lp
RACA_lp
PACA_lp
99
sole_src
done
custrqst
14
custque
redo
legal_rw
KO_rw
RACA_rw
PACA_rw
more_ARP


CMA
CS
cust
legal
KO
RACA
PACA
mgr
competiv
comp
approval
sole_src
done
custrqst
redo
rework
more_ARP

A complex software production process:

a *decomposition-precedence* relationship view
(19 levels of decomposition, 400+ tasks)



W. Scacchi, Experience with Software Process Simulation and Modeling, *J. Systems and Software*, 46(2/3):183-192,1999.

Computer-supported cooperative organizational learning environment

- Supports process modeling, simulation, re-enactment, and redesign.
- Supports capture, linkage, and visualization of ongoing group communications of developers, users, field researchers, and others
- Supports graphic visualization and animation of simulated/re-enacted processes, similar to computer game capabilities
- *Goal*: online environment that supports continuous organizational learning and transformation

Software production business models

- Custom software product engineering
- Agile production
- Revenue maximization
- Profit maximization
- Market dominance
- Cost reduction

Software production business models

- Custom software product engineering
  – Focus on Software Engineering textbook methods, with minimal concern for profitability
- Agile production
  – Focus on alternative development team configurations and minimal documentation, hence cost reduction
- Revenue maximization
  – Focus on stockholder value and equity markets, hence margin shrinkage in the presence of competition

Software production business models

- Profit maximization
  - Focus on developing and delivering reusable software product-lines; avoid one-off/highly custom systems
- Market domination
  - Focus on positioning products in the market by comparison to competitors; offer lower cost and more product functionality; continuous feature enhancement
- Cost reduction -- Open source software
  - Focus on forming internal and external consortia who develop (non-competitive) reusable platform systems; offer industry-specific services that tailor and enhance platform solutions

Questions?