Anna Kraus, Christopher Praas, Wiebke Rochler

Versionsverwaltung

	git - verteilte Versionsverwaltung	svn (Subversion) - zentralisierte Versionsverwaltung	Mercurial - verteilte Versionsverwaltung
Vorteile	• lokale Datenversionen (lokales Arbeiten ohne zentralen Server möglich) • raffinierte Mechanismen zum Zusammenführen der Änderungen • binäre Suche nach Fehlern die in vorherigen Versionen noch nicht da waren • Synchronisierung über HTTPS/SSH oder eigenem Protokoll • benötigt kein zentraler Server • Änderungshistorie komplett in Repository und Arbeitskopien enthalten • Strukturen leerer Verzeichnisse werden verworfen (kein Inhalt) • Absicherung beim Ausfall/Verlust des Hauptrepositories • schnell und effizient	• zentrale Verwaltung • pfadbasierte Zugriffsberechtigungen • besser bei vielen großen Binär-Dateien (Vergleich zu git) • Strukturen leerer Verzeichnisse werden vollständig aufgezeichnet	• lokale Kopie des Repositorys (gleich mit git) • bei großen Binären Dateien schneller als git • Benutzerfreundlichkeit und Verfügbarkeit • besser Strukturierte und verstehbare Befehle zum Synchronisieren
Nachteile	• aneinander kommen von mehreren Nutzern durch gleichzeitiges Arbeiten	• benötigt zentralen Server • zwei Nutzer können nicht gleichzeitig an derselben Datei arbeiten • Änderungshistorie nur im Repository komplett • Arbeitskopien erhalten nur neuste Version • bei jedem Zugriff wird Netzwerkanbindung benötigt • Ausfall/Verlust des Hauptrepositorys möglich	• mehr Befehle zum Zusammenfassen neuer Dateien • langsamer/nicht so effizient wie git

Wir entscheiden uns für git, da es verbreiteter ist, man also mehr Hilfestellungen dazu findet. Des Weiteren ist es sehr schnell und effizient.

Tool für Prototyping der Benutzerschnittstelle

	NinjaMock	Moqups	Pencil	Papier
Vorteile	• voller Funktionsumfang aller Designvorlagen bei kostenloser Nutzung • einfach & unkompliziert • benutzerfreundliche Oberfläche • gleichzeitige Bearbeitung durch mehrere Nutzer	• gut für kleine & mittlere Projekte • gleichzeitiges Bearbeiten möglich • in Google Drive integrierbar	• lokales Speichern möglich • Diagramme direkt im Programmfenster erstellbar	• schnellere Handhabung
Nachteile	• English	• lange Ladezeiten bei großen Projekten • begrenzte Auswahl von Elementen und Icons • unbeschränkte Version kostenpflichtig	• keine gute Dokumentation • eingeschränkte Nutzbarkeit • umfärben nicht möglich	• weniger professionell • nicht interaktiv

Wir entscheiden uns für NinjaMock, weil uns die Handhabung am einfachsten erschien und uns seine benutzerfreundliche Oberfläche überzeugt hat.

IDE/Editor

	Eclipse	IntelliJ IDEA	NetBeans
Vorteile	• sind bereits in Eclipse eingearbeitet • Erweiterbarkeit (Vielzahl an Plug-Ins) • mehr Erweiterungen & Dokumentationen	• Plug-In kann Funktionalität im Editor hinzufügen • zuverlässige Maven-und Git-Unterstützung • sehr schnell • Content Assist • Plug-Ins funktionieren zuverlässig untereinander • sehr schneller Support bei gefundenen Bugs & Wunsch Features	• Maven-Integration • Git-Integration • schreiben eigener Plug-Ins mit wenig Wissen in Java/Swing
Nachteile	• Maven-Unterstützung unzuverlässig • mehrere Plug-Ins funktionieren nicht zuverlässig untereinander	• Freeware hat nicht alle Features	• langsamer • basiert auf Swing • Debugging unübersichtlich

Wir entscheiden uns für Eclipse, da wir hier bereits eingearbeitet sind und schon wissen, wie wir welche Funktionen nutzen können. Des Weiteren bietet es viele Möglichkeiten der Erweiterung.

UML-Tool - Topcased

Wir entscheiden uns für Topcased, da wir darin bereits eingearbeitet sind und es sowieso die für Topcased benötigte IDE (Eclipse) nutzen.

Test-Automatisierung

	JUnit	TestNG
Vorteile	• kein unnötiger Code • zielgerichtetes „Coden“ • schnelle Überprüfung des Codes • Trennung von Code und Tests • Wiederverwendbarkeit von Tests • Übersichtlichkeit beim Testen	• von Eclipse unterstützt • erweitert JUnit um neue Funktionalitäten • Testen hinsichtlich Nebenläufigkeit möglich (Tests in parallelen Threads) • flexible Gruppierung der Tests und Testmethoden • Testmethoden mittels Parameter steuerbar • kurze Notation
Nachteile	• zusätzliche Methoden für Tests • unterbricht den Programmierfluss • abwägen zwischen Code und Tests	• anfängerunfreundlich

Wir entscheiden uns für JUnit, da dies weiterverbreitet ist (man also mehr Hilfe etc. dazu findet). Außerdem hat uns das Argument der schnellen Überprüfung des Codes und die Übersichtlichkeit überzeugt.

Taskmanagement/Bugtracker/Ticketsystem

Wir entscheiden uns für Bugzilla, da es schnell ist und einfach zu nutzen. Auch die erleichterte Kommunikation im verteilten Team überzeugt uns zur Nutzung.

	Bugzilla	Mantis	Fossil
Vorteile	• weit verbreitet • läuft auf jedem Betriebssystem • schnelle, einfache Fehlersuche • webbasiert • Übersichtlichkeit • auch für größere Projekte durch bessere Strukturierung geeignet • Benachrichtigung per E-Mail • erleichterte Kommunikation bei verteilten Teams • einfach zu nutzen	• besitzt die Lizenz GNU General Public License(GPL) • Erweiterbarkeit • klar geschriebener Code • webbasiert	• kein Verlust von Verbindungen durch um konfigurieren • Bei "Umziehen" auf einen anderen Rechner lediglich Kopieren einer Datei • Source Code des Projekts, Wiki und Bugtracking kann offline zugegriffen, später wieder synchronisiert werden • SQLite für die Datenspeicherung
Nachteile	• zunächst sehr überladene Oberfläche (nutzerunfreundlich)	• begrenzte Funktionalität • für größere Projekte eher ungeeignet	• nicht sehr verbreitet • meist nur für kleine Projekte genutzt

Dokumentationstool

	Javadoc	Doxygen
Vorteile	• direkt für Java ausgelegt, funktioniert sehr gut • Dokumentation während des Programmierens • Code & Dokumentation kann im Quelltext stehen • HTML Basierte Dokumentation wird erstellt • in vielen Programmen bereits eingebunden durch die IDE	• Klassendiagramme für Hierarchien und den Kooperationskontext • mehr zusammenfassende Seiten • optionales Quell-Code-Browsing (vernetzt mit der Dokumentation) • zusätzlicher Tag-Support (z.b. "@todo") auf separaten Seiten • Generation von pdf/TeX/... (und einigen mehr) Dokumenten möglich • viele visuelle Anpassungen möglich • kann zusätzlich zu Javadoc verwendet werden (ohne Javadoc zu zerstören) • kann für mehrere Programmiersprachen genutzt werden • komplexe mathematische Formeln werden sehr gut dargestellt
Nachteile	• ignoriert unbekannte Tags • für jedes Package wird eine html Datei benötigt • nur mit weiteren Doclets kann in PDF/XML/... exportiert werden	• nicht direkt auf Java ausgelegt • einheitliche Sprache muss gewählt werden, da sonst Fehler entstehen • Einträge sind meist nur in Englisch verfügbar, andere Sprachen sind nur teilweise übersetzt

Wir entscheiden uns für Doxygen, da dieses Tool mehr Features bereitstellt. Zum Beispiel bietet es die Möglichkeit der Generation verschiedener Dokumenttypen. Außerdem ist es einfach in der Handhabung und schnell.

Obfuscator

	Proguard	JavaGuard
Vorteile	• unterstützt alle Class-Dateien von Java Version 1.1 bis 8 • graphische Assistent erleichtert das Optimieren von Java-Anwendungen • leichter zu erlernen, als alle Andere kommerzielle Obfuscatoren • weiterverbreitet • wird aktiv gewartet	• wenig bis keine Informationen dazu im Web (nicht mehr als aktiv einzuschätzen)
Nachteile	• -	• scheinbar viele Bugs

Wir entscheiden uns für ProGuard, da wir für dieses Tool deutlich mehr Informationen gefunden haben. Außerdem gibt es dazu eine Vielzahl positiver Bewertungen.

Build-Tool

	ant	maven	gradle
Vorteile	• Plattformunabhängig • flexibel, alles unter Kontrolle • speziell auf Java ausgelegt	• modellbasiert, deklarativer Ansatz • Nachladen noch benötigter Bibliotheken nach Bedarf • hoher Automatisierungsgrad • unterstützt Tests und Dokumentationen • für große Java Projekte geeignet • Lebenszyklus abgedeckt	• vereint gute Ansätze von Maven und Ant • ausdrucksstarke domänenspezifische Sprache (DSL) (kürzere/einfachere Syntax als bei XML-Deklarationen von maven) • Lebenszyklus abgedeckt • für große Java Projekte geeignet • flexibel
Nachteile	• bereits veraltet • JRE nicht ausreichend • XML-Problem (schwer lesbarer, langer Code) • manuelle Analyse bei Migration zu anderer IDE • Automatisch generierte Skripte noch länger und umständlicher • für kleinere Projekte geeignet	• XML-Problem (schwer lesbarer, langer Code) • umstrittene Abhängigkeitsverwaltung • Anpassung für Spezialfälle schwierig	• massive Auslastung des Zwischenspeichers • relativ langsam • spezifische Sprache (DSL) nicht überall gern gesehen, da Aufwand sich in Programmiersprache einzufinden

Wir entscheiden uns für Gradle, weil es kombiniert die guten Ansätze von Maven und Ant. Des Weiteren hat uns die übersichtliche und leicht verständliche Syntax von Gradle‘s DSL überzeugt.

Kollaborationstool - Google Drive

Wir arbeiten des Weiteren mit Google Drive, als Tool für die Zusammenarbeit im Team. es ermöglicht uns das lesen, bearbeiten und hochladen von Dateien, an denen wir alle arbeiten können.
Außerdem haben wir auf diese von jedem beliebigen Laptop, Smartphone, Tablet, ... Zugriff.