

# expect & dejagnu

# Übersicht

- Szenarien
- Anforderungen an die Tests
- Testtools und Testumgebungen
- Fazit
- Quellen

expect &  
dejagnu

# Szenarien

- Extreme Programming
- Wartung und Pflege

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Extreme Programming

- Ziel ist langlebige, agile Software
- Projekte werden in kleine, überschaubare Aufgaben zerlegt
- kleine Entwicklerteams (max. 12 Personen), meist arbeiten je 2 an einer Aufgabe
- Releases in kurzen Abständen (ca. 3 – 4 Wochen)

## expect & dejagnu

1	Szenarien
1.1	Extreme Programming
1.2	Wartung und Pflege
2	Anforderungen
3	Tools/Umgebungen
4	Bewertung
5	Quellen

# Extreme Programming (2)

- Vorteile:
  - Schnelle Produktentwicklung
  - „test-driven-development“
- Probleme:
  - ein detaillierter übergeordneter Planungsprozess fehlt
  - Modifikationen an bereits getestetem Code (Modulen) sind jederzeit möglich

## expect & dejagnu

- 1 Szenarien
  - 1.1 Extreme Programming
  - 1.2 Wartung und Pflege
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Wartung und Pflege

- Korrekturen, Fehlerbeseitigung
- Leistungsverbesserung
- Anpassung an neue Anforderungen bzw. Umgebungen (zB. Portierung)
- Erweiterung um neue Funktionen oder Schnittstellen

## expect & dejagnu

- 1 Szenarien
  - 1.1 Extreme Programming
  - 1.2 Wartung und Pflege
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Wartung und Pflege (2)

- Probleme:
  - durch Wartung (Behebung von Fehlern) und Pflege (Erweitern und Ändern) können unerwünschte Seiteneffekte in Modulen und an Schnittstellen auftreten

## expect & dejagnu

- 1 Szenarien
  - 1.1 Extreme Programming
  - 1.2 Wartung und Pflege
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Anforderungen an die Tests

- Planung und Vorbereitung
- Durchführung
- Auswertung

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Planung und Vorbereitung

- möglichst wenig Einarbeitung in ein Testtool bzw. eine Testumgebung
  - Trotz umfangreicher Testmöglichkeiten: einfache Definitionssyntax
- Wiederverwendbarkeit von Testroutinen
- gutes Verhältnis von Programmcode und Testcode

## expect & dejagnu

1	Szenarien
2	Anforderungen
2.1	Planung und Vorbereitung
2.2	Durchführung
2.3	Auswertung
3	Tools/Umgebungen
4	Bewertung
5	Quellen

# Durchführung

- möglichst automatisiert oder zumindest teilautomatisiert

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  - 2.1 Planung und Vorbereitung
  - 2.2 Durchführung
  - 2.3 Auswertung
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Auswertung

- Unterschiedliche Anforderungen:
  - maschinenlesbar, um Protokolle in Entwicklungsumgebungen einbinden zu können
  - für Menschen lesbar, um Entwicklungsprozesse effizient gestalten zu können

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  - 2.1 Planung und Vorbereitung
  - 2.2 Durchführung
  - 2.3 Auswertung
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Testtools und Testumgebung

- Tcl/Tk
- expect
- dejagnu
  - für Kommandozeilen-Programme
  - für graphisch-interaktive Programme

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

# Tcl/Tk

- Tcl – Tool Command Language
  - OS-unabhängige Skriptsprache von Sun Microsystems
  - kann Programme ansprechen, steuern, etc.
- Tk – Toolkit
  - stellt einen Interpreter zur Verfügung (wish), über den User-Interfaces gestaltet werden können
  - Programm- bzw. Objektzustände können abgefragt werden

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Tcl/Tk (2)

- Tcl kann direkt in Programme eingebunden werden
- es kann um AddOns erweitert werden
- Mit Hilfe des Toolkits kann für Kommandozeilenprogramme eine graphische Oberfläche erstellt werden, die mit dem Expect-AddOn gesteuert werden

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# excpect

- entwickelt von Don Libes
- Tcl/Tk-AddOn
  - <http://expect.nist.gov/>
- Name basiert auf dem Befehl „expect“, mit dem Ausgaben von der Kommandozeile gelesen werden

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Befehle in expect

- **spawn**
  - erzeugt einen neuen fernzusteuerten Prozess
- **expect**
  - liest eine Ausgabe von der Kommandozeile
  - Ausgaben können gefiltert werden
  - Reguläre Ausdrücke mit Option `-re` möglich (anstelle von `-gl`)

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.2.1	Befehle
3.2.2	Beispiel (ftp)
3.2.3	Beispiel (bc)
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Befehle von expect (2)

- **send**
  - simuliert Benutzereingaben
- **interact**
  - ermöglicht dem Benutzer ein Eingreifen in den Skriptablauf

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.2.1	Befehle
3.2.2	Beispiel (ftp)
3.2.3	Beispiel (bc)
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Beispiel für expect

## Steuerung eines ftp-Clients

```
# ftp starten und Eingabeaufforderung abwarten
spawn ftp
expect "ftp> "

# Verbindung herstellen
send "open ftp.gnu.org\r"

while 1 {
  expect {
    "Name *: "      { send "anonymous\r" }
    "Password: "   { send "email@domain.com\r" }
    "*ftp> "       break
    timeout        exit
  }
}

# binary-Befehl absenden
send "binary\r"
expect "*ftp> "

#an Benutzer abgeben
interact
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
    - 3.2.1 Befehle
    - 3.2.2 Beispiel (ftp)
    - 3.2.3 Beispiel (bc)
  - 3.3 dejagnu
    - 3.3.4 Beispiel (hello)
  
- 4 Bewertung
- 5 Quellen

# Regressionstests mit expect

- Szenario
  - Implementierung wurde geändert
  - Spezifikation ist gleichlautend
- Lösung: Regressionstest
  - ein gespeichertes Testskript wird noch einmal mit dem geänderten Programm/Modul ausgeführt

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.2.1	Befehle
3.2.2	Beispiel (ftp)
3.2.3	Beispiel (bc)
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Beispiel für Regressionstests mit expect

```
# starten und warten
spawn bc
sleep 1

# Testergebnisse codieren
proc pass {a} { puts "erfolgreich: $a" }
proc fail {a} { puts "fehlgeschlagen : {a}" }

# Testroutine
proc bc_test {expr result} {
    send "$expr\n"
    expect "$expr\r\n"

    expect {
        -gl "$result\r\n" { pass "$expr" }
        "*\r\n"           { fail "$expr" }
        timeout           { fail "timeout $expr" }
    }
}

# Testfälle
bc_test "2+2" "4"
bc_test "7*9" "63"
...
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
    - 3.2.1 Befehle
    - 3.2.2 Beispiel (ftp)
    - 3.2.3 Beispiel (bc)
  - 3.3 dejagnu
    - 3.3.4 Beispiel (hello)
- 4 Bewertung
- 5 Quellen

# Beispiel für Regressionstests mit expect (2)

- es können nun beliebig viele Testfälle hinzugefügt werden
- die Ergebnisse werden auf der Kommandozeile ausgegeben
- expect erlaubt teilautomatisierte Tests; die Auswertungen erfolgen manuell

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.2.1	Befehle
3.2.2	Beispiel (ftp)
3.2.3	Beispiel (bc)
3.3	dejagnu
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# dejagnu

- erweitert expect um Schnittstellen zur Testautomatisierung
- ist ein Framework für Regressionstests
- Realisation nach der POSIX 1003.3 – Norm; diese sieht 5 Testergebnisse vor

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.4 Beispiel (hello)
- 4 Bewertung
- 5 Quellen

# Testergebnisse (POSIX1003.3)

- **PASS**
- **FAIL**
- **UNRESOLVED**
  - Testergebnis kann nicht ausgewertet werden
- **UNTESTED**
  - Test wurde nicht ausgeführt; fungiert als Platzhalter
- **UNSUPPORTED**
  - Umgebung unterstützt Test nicht

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.1	Regeln
3.3.2	Framework
3.3.3	Beispiel (bc)
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# dejagnu Framework

- jedem Testausgang wird eine Ausgabe (pass, ...) zugeordnet
- Zusätzlich sind folgende Prozeduren zu definieren:
  - name\_load
  - name\_start
  - name\_exit
  - name\_version
- name wird durch den Programmnamen ersetzt

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.1	Regeln
3.3.2	Framework
3.3.3	Beispiel (bc)
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Beispiel für dejagnu

## Auswertung math. Strings

```
# Testfall laden
proc bc_load {arg} {
    # nichts weiter
}

#Programm starten
proc bc_start {} {
    global spawn_id
    spawn "bc"
    sleep 1
}

#Programm beenden
proc bc_exit {} {
    send "quit\n"
}

#Programmversion ermitteln
proc bc_version {} {
    return "unknown"
}
```

--> --> -->

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
  
- 4 Bewertung
- 5 Quellen

# Beispiel für dejagnu

## Auswertung math. Strings (2)

```
                                <-- <-- <--  
  
#Auswertung  
proc bc_test {expr result} {  
    send "$expr\n"  
    expect "$expr\r\n"  
  
    expect {  
        -gl "$result\r\n" { pass "$expr" }  
        "*\r\n"           { fail "$expr" }  
        timeout           { fail "timeout $expr" }  
    }  
}  
  
#abschließend: Programm starten  
bc_start
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
  
- 4 Bewertung
- 5 Quellen

# dejagnu Framework (2)

- Bestandteile eines Tests:
  - Steuerungsskript (vorherige Seite)
  - Testfälle
    - in einzelnen Skripten
    - Skripte enden auf .exp und liegen in einem Unterverzeichnis von name.suffix (name = Programmname; suffix = beliebig, meist „test“)
- Testfälle werden vom dejagnu-Framework automatisch geladen und ausgeführt

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.1	Regeln
3.3.2	Framework
3.3.3	Beispiel (bc)
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# dejagnu Framework (3)

- Testablauf:
  - Es gibt ein Hilfsprogramm „runtest“
    - `runtest --all --tool name`
  - `name` ist der Programm-/Testname

(eines der Testskripte)

```
# einige Testfälle
bc_test "2+2"    "4"
bc_test "7*9"    "63"
bc_test "3-7"    "-4"
bc_test "9/3"    "4"    // <- künstl. Fehler
...
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
- 4 Bewertung
- 5 Quellen

# Beispiel für dejagnu

## Auswertung math. Strings (2)

(Ausgabe)

```
$ runtest --all --tool bc
Test Run by clessner on Mon May 24 10:00:00 2004
Native configuration is sparc-sun-sunos4.1.4

    === bc tests ===

Running ./bc.test/test1.exp ...
PASS: 2+2
PASS: 7*9
PASS: 3-7
FAIL: 9/3

    === bc summary ===

# of expected passes      3
# of unexpected failures  1

$ _
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
  
- 4 Bewertung
- 5 Quellen

# dejagnu Framework (4)

- batch-Tests können vollautomatisiert durchlaufen
- Es gibt eine detaillierte Beschreibung der Eingaben und Ausgaben (Fehler), sowie eine Zusammenfassung
- Bei geändertem Code kann der Test einfach wiederholt werden

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.1	Regeln
3.3.2	Framework
3.3.3	Beispiel (bc)
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# dejagnu für graphisch-interaktive Programme

- Programme, deren graphische Oberfläche mit dem Toolkit für Tcl erstellt wurden, lassen sich fernsteuern
- der wish-Interpreter stellt dazu eine Schnittstelle bereit
- Zustände der Widgets können gelesen und gesetzt werden (d.h. es wird die Interaktion eines Benutzers simuliert)

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.1	Regeln
3.3.2	Framework
3.3.3	Beispiel (bc)
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Beispiel für dejagnu

## „Hello World“ mit Tcl/Tk-GUI

(Programmcode)

```
button .b -text "Hello, world!" -command {puts
    "Hello Test" }
pack .b
```

(Testprogramm)

```
proc hello_load { arg } {
    # nichts
}

proc hello_start {} {
    global spawn_id
    spawn "wish"
    expect "% "
    send "source hello.tcl\n"
}
```

--> --> -->

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
  
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
  
- 4 Bewertung
- 5 Quellen

# Beispiel für dejagnu

## „Hello World“ mit Tcl/Tk (2)

```
proc hello_exit {} {
    send "exit\n"
}

proc hello_version {} {
    return [info tclversion]
}

# Testroutine
proc hello_test {cmd result } {
    send "$cmd\n"
    expect {
        "$result\r\n" { pass "$cmd" }
        "*% " { fail "$cmd" }
        timeout { fail "timeout $cmd" }
    }
}

# abschließend: Programm starten
hello_start
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
- 4 Bewertung
- 5 Quellen

# Beispiel für dejagnu

## „Hello World“ mit Tcl/Tk (3)

(Testskript)

```
hello_test ".b cget -text" "Hello, world!"  
hello_test ".b invoke"     "Hello Test"
```

(Ausgabe)

```
$ runtest --all --tool hello  
Test Run by clessner on Mon May 24 10:00:00 2004  
Native configuration is sparc-sun-sunos4.1.4  
  
    === hello tests ===  
  
Running ./hello.test/test1.exp ...  
PASS: .b cget -text  
PASS: .b invoke  
  
    === hello summary ===  
  
# of expected passes          2  
  
$ _
```

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
  - 3.1 Tcl/Tk
  - 3.2 expect
  - 3.3 dejagnu
    - 3.3.1 Regeln
    - 3.3.2 Framework
    - 3.3.3 Beispiel (bc)
    - 3.3.4 Beispiel (hello)
- 4 Bewertung
- 5 Quellen

# dejagnu für graphisch-interaktive Programme (2)

- Vorteile:
  - das komplette Programm kann vom Framework gesteuert werden
- Probleme:
  - es wird nur der interne Zustand analysiert; interne Änderungen (z.B. Variablennamen) wirken sich auf das Ergebnisprotokoll aus
  - die eigentliche Oberfläche kann nur schwer getestet werden (z.B. ob Buttons im Sichtbereich liegen, etc)

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
3.1	Tcl/Tk
3.2	expect
3.3	dejagnu
3.3.1	Regeln
3.3.2	Framework
3.3.3	Beispiel (bc)
3.3.4	Beispiel (hello)
4	Bewertung
5	Quellen

# Bewertung/ Fazit

- Tcl/Tk – expect – dejagnu – Framework ist gut für batch-orientierte Programme benutzbar
- Automatisierte Regressionstests werden gut unterstützt
- „weiche“ bzw. „logische“ Einschätzung der Ausgabeergebnisse möglich
  - „Ergebnis von  $3*4$  ist 12“ wird als ebenso richtig eingestuft wie „ $3*4 = 12$ “ (Filter: „expect \*\$result“)

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
4	Bewertung
5	Quellen

# Bewertung/ Fazit

- dejagnu ist ein einfaches Framework, der schnell angepasst werden kann (zB. durch das Hinzufügen neuer Testfälle)
- Ausgaben können maschinenlesbar gestaltet werden
- graphische Schnittstellen können bedingt getestet werden

## expect & dejagnu

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
4	Bewertung
5	Quellen

# Quellen

- Bücher:
  - Helmut Balzert „Lehrbuch der Softwaretechnik“, Bd. I, 2.Aufl.
  - Handout, Kapitel 10 „Software-Tests mit DEJAGNU“
- Dokumentationen:
  - <http://www.frankwestphal.de/Extrem>
  - <http://www.clug.de/vortraege/expect>
  - <http://www.delorie.com/gnu/docs/de>
  - <http://www.gnu.org/software/dejagnu>

## expect & dejagnu

- 1 Szenarien
- 2 Anforderungen
- 3 Tools/Umgebungen
- 4 Bewertung
- 5 Quellen

- Sourcecodes/Binaries:
  - <http://expect.nist.gov/>
  - <http://bmrc.berkeley.edu/people/cha>

1	Szenarien
2	Anforderungen
3	Tools/Umgebungen
4	Bewertung
5	Quellen

# Fragen ?

expect &  
dejagnu