Kosten und Risiken der Softwareentwicklung reduzieren durch automatische Software-Produktion

KooperationsForum Computer Hard- und Software

Technologietransfer aus der Raumfahrt

Berlin

24.10.2001

Dr. Rainer Gerlich
BSSE System and Software Engineering
Auf dem Ruhbühl 181
D-88090 Immenstaad
Phone: +49/7545/91.12.58
Mobile: +49/171/80.20.659
Fax: +49/7545/91.12.40
e-mail: gerlich@t-online.de
URL: http://home.t-online.de/home/gerlich/

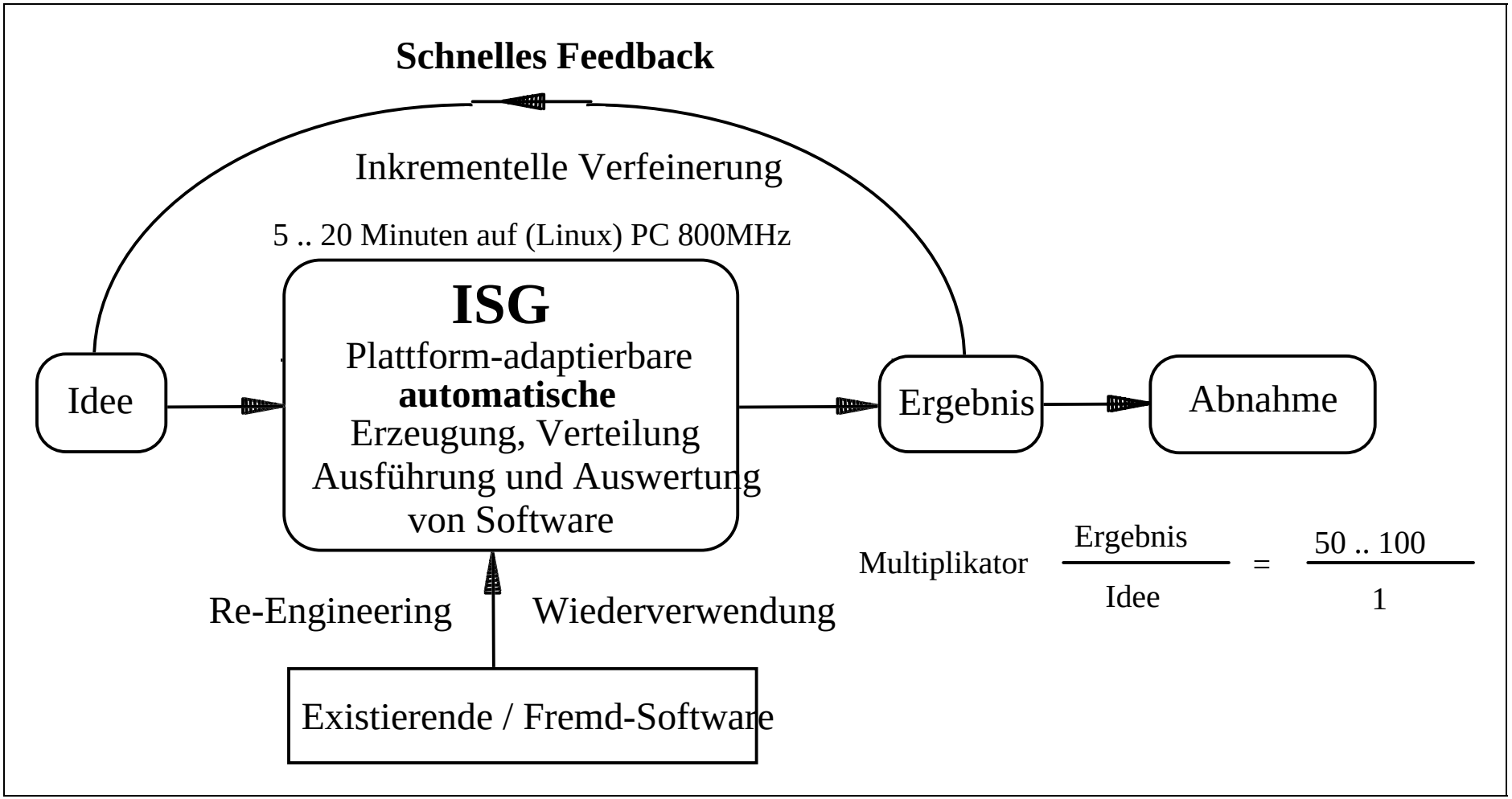
BSSE System and Software Engineering

Automation - Lernen von anderen Bereichen

- Übergang von manueller auf automatisierte Produktion
  - Fahrzeuge, Unterhaltungselektronik, Haushaltsgeräte, Lebensmittel .....
- Vorteile
  - vollständig automatiserter Produktionsprozess — Aufwand ↓
  - schnelle Verfügbarkeit des Produktes — Time-To-Market ↓
  - definerter und reproduzierbarer Produktionsablauf — Qualität ↑
- Vorgehensweise
  - bekannte Entwicklungsaktivitäten auf Automation ausrichten
  - neue Aktivitäten manuell durchführen und später automatisieren

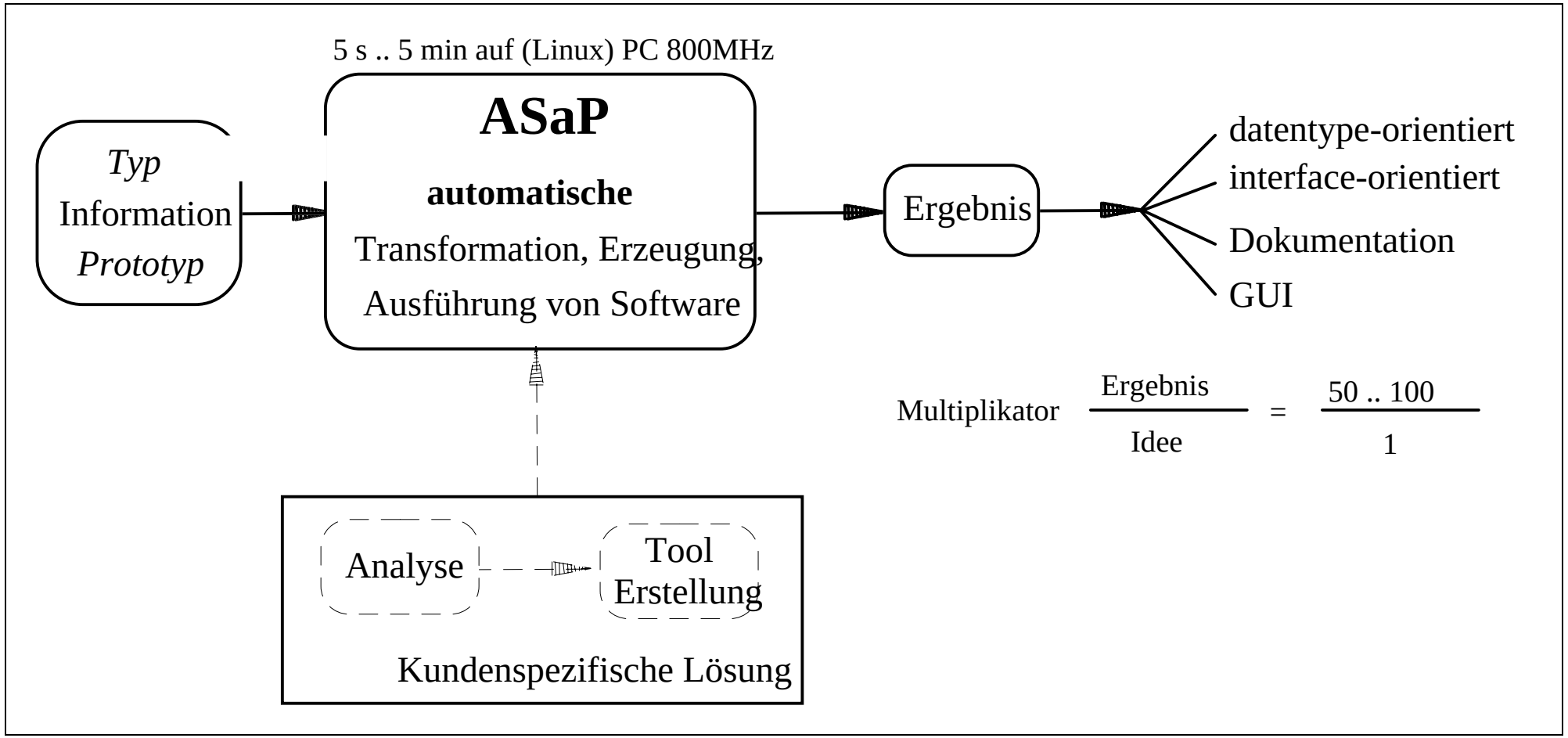
Beispiel 1: Der ISG-Entwicklungszyklus

ISG = Instantaneous System and Software Engineering



Beispiel 2: Der ASaP-Entwicklungszyklus

ASaP = Automated Software Production



Softwareentwicklung - Geschützt gegen Rationalisierung ?

- Informelle Vorgehensweise
  - Aktivitäten, um Probleme zu verstehen, aber nicht zu lösen
    "Use cases": Systemprototyp, aber dann davon unabhängige Implementierung
  - Konzentration mehr auf Produktion, weniger auf Test und Qualität
- Akzeptanz der Probleme
  - Software ist komplex, daher sind Fehler unvermeidbar
  - Software ist komplex, daher ist Softwareentwicklung teuer
  - Software ist komplex, daher ist keine Rationalisierung möglich
- Akzeptanz der Situation
  - Einsatz von Softwareentwicklungstools = bereits höchste Rationalisierungsstufe
  - Festhalten an phasen-orientierten Entwicklungsabläufen (Wurzeln in 196x)
  - Einführung von neuen Konzepten, die nur Teilautomation zulassen
    objekt-orientiert: Klassen und Vererbung decken strukurelle Änderungen nicht ab

Softwareentwicklung und Kostenanalyse

- Produktivität
  - Wem sind allgemeine Produktivitätszahlen bekannt?
  - Wer kennt seine eigene Produktivität?
  - Wer hat ein Programm zur Produktivitätssteigerung?
  - Wer hat in der Informatikausbildung etwas über Produktivität gehört?
- Return-Of-Investment (ROI)
  - Wer kann Zahlen nennen, ob und wann Investitionen in Ausbildung und Infrastruktur einen Ertrag gebracht haben?
  - Wer hat solche Zahlen im eigenen Haus zur Verfügung?
- Time-To-Market
  - Wer hat eine Strategie, um die Entwicklungszeit zu verkürzen?
  - Wer konnte durch gezielte Maßnahmen bereits die Entwicklungszeit verkürzen?
- Aufwandsanalyse
  - Wer analysiert den Softwareentwicklungsprozess auf Schwachstellen?
  - Wer gibt Produktivitätssteigerungen vor zur Verbesserung der Wettbewerbssitutation?

Softwareentwicklung und Risikoanalyse

- Entwicklungsrisiko
  - Wer wendet einen systematischen Ansatz an, um Risiken frühzeitig zu erkennen?
  - Wer konnte durch Simulation eines Softwaresystems das Risiko senken?
  - Wer konnte durch Simulation das Risiko einschätzen bzw. begrenzen?
- Tool Support
  - Wieviel Prozent des Aufwandes des Entwicklungszyklus decken Tools ab?
  - Wie hoch ist die Wahrscheinlichkeit, dass ein von einem Tool akzeptierter Entwurf (oder eine Spezifikation) ohne jegliche weitere Änderung / Nacharbeit erfolgreich umgesetzt werden kann?
  - Wie hoch ist der (prozentuale) Aufwand, um die mit einem Tool erarbeitete Lösung in die endgültige Version umzusetzen?
- Softwareentwicklungsmethodik
  - Mit welcher Entwicklungsmethode wurden rechzeitig Risiken erkannt?
  - Wer verfügt über entsprechende Zahlen?

Softwareentwicklung und Qualitätssicherung

- Testabdeckung
  - Wer kennt den Austestungsgrad seiner Software?
  - Wer wendet "Stress Testing" an?
  - Wer testet gezielt auf Daten-, Eingabe- und zeitliche Fehler und auf Überlast?

- Qualitätsverbesserung (TQM)
  - Wer setzt eine Strategie zur Verbesserung der Softwarequalität ein?
  - Wer kann die erreichte Qualitätsverbesserung quantifizieren?

    z.B. durch absolute/relative Zahlen über aufgetretene Fehler nach Auslieferung, Höhe der Kosten der Fehlerbeseitigung

Software-Entwicklungsmethoden

- Ziele des Einsatzes
  - geringerer Aufwand
  - kürzere Entwicklungszeit
  - weniger Risiko
- Wasserfall- und V-Modell
  - viele manuelle Schritte zwischen den Phasen erforderlich — Aufwand ↑
  - sequentieller Ablauf / Unterauftragnehmer-Hierarchie — Zeit ↑
  - viel Papierarbeit am Anfang, zu einem späten Zeitpunkt: ausführbares und integrationsfähiges System — Risiko ↑
- Spiralmodel, Rapid Prototyping
  - später nicht brauchbare Zusatzaktivitäten — Aufwand ↑
  - Zusatzaktivitäten — Zeit ↑
  - Prototyping
    - repräsentativ (vor Vollendung kaum einschätzbar) — Risiko ↓
    - nicht-repräsentativ — Risiko ↑

Ansätze zur Aufwandsreduktion und Risikominimierung

- Simulation und Analyse
  - Aufwand
    Reduktion durch Vereinfachung: genügend repräsentativ?
  - Risiko
    gut bei kontinuierlichen Vorgängen (Mechanik, Elektronik)
    keine Vorhersage möglich bei diskreten Vorgängen (ereignisgesteuert)
- Wiederverwendung
  - Aufwand
    bei interner / eigener Wiederverwendung / Quellcode √
    bei Fremdsoftware / COTS: nicht einschätzbar
  - Risiko
    bei interner / eigener Wiederverwendung / Quellcode begrenztes Risiko
    bei Fremdsoftware / COTS unbekanntes Risiko
- Verzicht auf Entwicklungskonzept
  - "quick and dirty"
  - "Extreme Programming" (eCommerce)

Automation und Softwareentwicklung

- sinnvoll bei Software ?
  - Softwaresystem = Unikat ?
  - Softwareerstellung = schöpferischer Prozess ?
    aber: "1% inspiration, 99% transpiration"
- Einführung von Konstruktionsregeln und Schnittstellen
  - Identifikation generischer Regeln zur breiten Abdeckung durch Automation
  - Auswahl eines bestimmten Konstruktionsprinzips ohne Beschränkung der Allgemeinheit
  - nur die Auswahl eines Konstruktionsprinzips führt zu Verbesserungen
- geeignete Strukturierung der Systemdefinition
  - wenn Automatisierung, dann vollständig
  - noch notwendige manuelle Eingriffe senken erheblich die Effizienz des Konzeptes
- genügend Flexibilität
  - Abdeckung möglicher Änderungen (Kommunikationsstruktur, Topologie, Plattform etc.)
  - leichte/effiziente Einbindung von manuell erstellter Software (effiziente Anfangsphasen)
- Synergie durch Automation
  - Kombination von Systemgenerierung und -test, Ausführung, Verifikation, Validierung
  - Kombination mit Dokumentation, GUI-Erstellung

Automation und Qualität

- automatisiertes Prozessmodel
  - keine manuellen Eingriffe im Gesamtablauf
  - Herstellungsprozess vollständig kontrollierbar (im Sinne von ISO 9000 ...)
  - Qualitätsüberwachung automatisierbar
- Einfluss auf Qualität
  - Möglichkeit der objektiven, automatischen Qualitätskontrolle
  - Möglichkeit zur Verbesserung des Produktes
- je breiter und vielfältiger der Einsatz von Software,
  desto wichtiger ist die Qualität
- Automation verbindet Effizienz- mit Qualitätssteigerung

ASaP - Automated Software Production

- ISG = Instantaneous System and Software Generation
  - Infrastruktur für verteiltes und/oder Echtzeit-System
  - automatische Integration von automatisch erzeugter Software mit manuell erzeugter / vorhandener Software
  - Ergebnis schrittweiser Erhöhung des Automatisierungsgrades über eine Reihe von Projekten aus verschiedenen Gebieten

    Raumfahrt, Telekommunikation, Flugsicherung, Kraftwerktechnik
  - Automatisierung der Produktion verteilter und/oder Echtzeitsysteme
    - **Information anliefern → Produktionsprozess starten → Testbericht lesen**
- ASaP
  - ISG und/oder automatische Erzeugung sequentieller, anwendungsspezifischer Software inkl. Dokumentation
  - Ergebnis der Identifikation weiterer Automatisierungspotenziale zusammen mit Kunden
  - konsequente Umsetzung in allgemein brauchbaren Ansatz
    - datentyp-orientiert: Generierung einer Funktionsmenge zu einer Menge von Typen
    - interface-orientiert: Adaption / Konvertierung von Schnittstellen, PSprachkonvertierung
    - Dokumentation: automatische Generierung von Dokumentation inkl. Informationmsfilterung (RTF, MS-Word, pdf)
    - GUI: zukünftige Aktivität

Prinzipielle Vorgehensweise

- ISG
  - Identifikation der Systembestandteile
    über ISG-Systembeschreibungsmerkmale: Prozesse, Status, Messages
  - automatischer Ablauf: Systemgenerierung, Testfallgenerierung, V&V, Berichterstellung
  - Iterationen zur Systemstruktur und Funktionalität bis zur endgültigen Version
  - volle Integrationsfähigkeit und Plattform-Portabilität / -Unabhängigkiet von Anfang an
- ASaP
  - Einsatz vorhandener ASaP-Werkzeuge
    Funktionsgeneratoren, Dokumentationsgeneratoren, Testgeneratoren, GUI Builder
  - Identifikation der Kostentreiber und der Risiken mit Kunden
    - Erstellen weiterer allgemein einsetzbarer Generatoren
    - Erstellen kundenspezifischer, wiederverwendbarer Generatoren
- ISG und ASaP
  - Automation bis zur Grenze des Machbaren in der aktuellen Iteration
  - leichte und reproduzierbare Einbindung von anderem Quellcode (manuell, andere Tools)
  - Erhöhung des Automatisierungsgrades in der nächsten Iteration

ASaP und der Erstellungsaufwand

- Minimierung von Entwicklungskosten und -zeit erfordern
  - Analyse des Aufwandes $f(x)=a + b \cdot x + c \cdot x^2 + d \cdot x^3 + \mathbf{...}$
  - Identifizierung der Kostentreiber
    höhere Potenzen von x (Menge) und große Koeffizienten
  - konsequente Automatisierung der kostentreibenden Prozessanteile
    - vollständiger Ausschluss manueller Eingriffe

      auch nur ein kleiner Teil noch verbleibender manueller Tätigkeiten
      verhindert die erreichbaren erheblichen Einsparungen an Kosten und Zeit
    - Anwendung von "Konstruktionsregeln", Erzeugen von "Generatoren"

      Erschließung von Bereichen, die durch Programmiersprachen,
      nicht abgedeckt werden (auch nicht durch OOP / AOP)
- ASaP / ISG
  - setzen die begrenzte Entwicklerkapazität frei für "kreative" Tätigkeiten
  - binden nicht die wertvolle Arbeitskraft für "nicht-kreative" Tätigkeiten
  - automatisieren kostenintensive Tätigkeiten durch Ableiten von Information von nicht-kostenintensiven Einstiegs-Schnittstellen

Beispiele für Kostenreduktion (1/2)

- Reduktion von f(x)=x auf f(x)=const (Unabhängigkeit von Menge)
  - Schnittstelle zwischen Programmiersprachen

    Anbindung von C Funktionen an (Script-) Interpretersprache

    Erzeugung der benötigten Funktionalität aus vorhandener Information der C-Prototypen
    - bei Beginn des Einsatzes von ASaP: ca. 130 Funktionen, jetzt ca. 450 Funktionen
    - komplexe Interfaceanpassung:

      Parameterübergabe (Beschreibung, Parameter-Alignment, Konvertierung)

      Funktions-Prototypen und Typdeklarationen
    - Erstellung von Demo- und Test-Scripts

      Schnittstelle zwischen Script und Funktionsaufruf (Umsetzung der Eingabe)

      Erzeugung möglicher Eingaben (interaktive Parametervorschläge)
    - für neue Version oder bei neuen Funktionen fällt kein Aufwand für die Erstellung des Interfaces und der Script-Umgebung mehr an
    - neue Version der Software ist nach ca. 2 Minuten auf Knopfdruck verfügbar
    - keine neuen Tests erforderlich für neue Funktionen
    - Erstellen des Benutzerhandbuches (RTF, MS-Word, pdf)
      ca. 900 Seiten mit extrem vielen Links: ca. 1 Minute auf PC-800MHz

Beispiele für Kostenreduktion (2/2)

- Reduktion von $f(x)=b \cdot x^2$ auf $f(x)=a \cdot x$

  Reduktion auf linearen Kostenanstieg mit a << b

  - Definition von Punkt-zu-Punkt-Verbindungen in einem TCP/IP Netzwerk

    Aufwand = $o(n^2)$ wobei n = Anzahl der Knoten

    - wenn die n Knoten bekannt sind, kann die Software für die Verbindungen automatisch erstellt und getestet werden $o(n^2) \rightarrow o(n)$

- Beispiel für ineffektive Automatisierung: Testen

  - wenn das Testen Information erfordert, die aus dem erstellten Quellcode manuell abgeleitet werden muss
  - besser:

    automatische Ableitung des Quellcodes aus Anwendervorgaben bei gleichzeitiger Ableitung der Testinformation

ISG/ASaP-Anwendung für ISS

- Experiment für Internationale Raumstation ISS
  - komplexes Experiment
    Kristallzüchtung in Schwerelosigkeit, viel prozesstechnische Peripherie
  - verteilte Echtzeitinfrastruktur mit ISG (zwei Rechner, 40 $\rightarrow$ 30 Prozesse)
  - ASaP: verteilte Datenbank, Datenerfassung, Kalibrierung,
    Grenzwertüberwachung, Telemetrie
  - Generierungszeit: ca. 10 .. 20 Minuten auf PC-800MHz (vollständige Generierung)
  - Auslieferung Sommer 2000, seit Frühjahr 2001 in Systemintegration
  - kontinuierliche Analyse des Speicher- und Zeitbedarfs
  - durch einfache Restrukturierung der verteilten Datenbank (Aufwand ca. 1/2 Tag)
    für ca. 600 Datenelemente konnte Performance um ca. 30% gesteigert werden

    geschätzter Aufwand bei konventioneller Vorgehensweise: ca. 1 MM
    inkl. 1 Monat Ungewissheit, ob wirklich erfolgreich
  - Kundenkommentar:
    - nur mit ISG / ASaP war Komplexität zu bewältigen
    - erhebliche Zeitersparnis durch automatische Portierung zwischen verschiedenen Plattformen während Integration inkl. Wechsel zwischen 1- und 2-Rechner-System
    - Änderung an verteilter Datenbank getestet auf einem Rechner, sofort fehlerfrei lauffähig auf 2-Rechner-System

Synchrones Verteiltes System

- Studie im Rahmen von ESPRIT-Projekt: Kraftwerkstechnik
  - verteiltes System mit redundanten Komponenten, 16 Prozessoren, "Voting"
  - Echtzeit-Infrastruktur mit ISG
  - Elemente für synchrone Datenverarbeitung: asynchrones Speichern, synchrones Lesen inherent unterstützt durch ISG (Option)
  - Integration von (teil-automatisch) synchroner C-Software in ISG-erzeugte Infrastruktur
  - Erzeugung der Testumgebung zum Untersuchen des Einflusses von
    - "Time Jitter" auf Voter-Entscheidungen
    - fehlerhaften Daten auf Voter-Entscheidungen
  - Ergebnis:
    - starke Abhängigkeit von Time Jitter
    - teilweise sensitiv auf Datenfehler
  - Generierungszeit (vollständige Systemgenerierung für eine Iteration):

    ca. 10 Minuten auf PC-800MHz

Erzeugen von Funktionsmengen mit ASaP

- Binäre Datenkonvertierung
  - generischer Ansatz für Konvertierung zwischen "Little Endian" und "Big Endian"
  - Generierung für jeden (kompletten) Satz von anwender-definierten Datentypen durch Vorgabe eines generischen Funktionsskeletts
    - Hin- und Rückkonvertierung
    - Initialisierung, Testausführung und Test auf Korrektheit
  - Generierungszeit: ca. 5 s auf PC-800MHz
- interaktive Testumgebung für Funktionen
  - zur Zeit ca. 450 Funktionen
  - Konvertierung der Benutzereingaben (Texte) in erforderliche Datentypen der Parameter
  - kontext-relevante Vorschläge für Parameter (optional)
  - funktionsübergreifende Parameterversorgung

    kumulative Inputs, Output $\rightarrow$ Input für nächste Funktion
  - Generierungszeit: ca. 20 s auf PC-800MHz

Weitere Anwendungen

- Signifikante Verkürzung der Reaktionszeit
  - Aufgabe: Erstellen einer kundenspezifischer Konfiguration
  - Dauer: zur Zeit ca. 1 Monat
  - Ziel: mit ASaP höchstens zwei Stunden
- Optimierung von Verarbeitungsalgorithmen
  - Aufgabe: Generierung von definierten Testmustern
automatischer Vergleich der Ergebnisse mit Ausgangswerten
  - Ziel: mit ASaP automatisch Testmuster erzeugen und Ergebnisse auswerten
- Zuverlässigkeitsanalyse und Erhöhung der Zuverlässigkeit
  - Aufgabe: Software muss innerhalb eines kurzen, vorgegebenen Zeitfensters zuverlässig eine bestimmte Anzahl von Funktionen ausführen
  - Ziel: mit ASaP durch automatische Tests Zuverlässigkeitsprofil erstellen, Schwachstellen erkennen und Nachweis für erforderliche Zuverlässigkeit erbringen

Verfügbarkeit

- ISG
  - Unix / C (SunOS und Linux)
    - Sparc (SunOS 5.5, 5.7), Intel/PC (Linux 2.1.111, 2.2.17)
  - VxWorks / C
    - Sparc und Intel/PC, Version 5.3
  - heterogener Betrieb möglich
    - beliebige Kombination aus Prozessor und OS zu jedem Zeitpunkt
    - Ausführung auf der Plattform, die gerade verfügbar ist
  - MS-Windows: in Vorbereitung
  - "bare machine": möglich
- ASaP
  - C
  - andere Sprachen möglich

Einsatz von ISG und ASaP

- ISG und ASaP
  - als Produkte verfügbar
  - kundenspezifische Weiterentwicklung möglich
  - empfehlenswert: erster Einsatz in Kooperation
- Anwendung der Erfahrung auf kundenspezifische Infrastruktur
  - Analyse der Umgebung zusammen mit dem Kunden
  - Entwicklung kundenspezifischer ASaP-Software
  - Integration in Betriebsablauf
  - Training
- Anwendungsgebiete
  - verteilte Systeme, Echtzeitsysteme
  - daten- und interfaceorientierte Optimierungen
  - Testfallgenerierung inkl. Stresstesting und Fehlereinspeisung, V&V
  - Qualitätsanalyse, Berichterstellung, GUI Builder

Vorgehensweise

- Ist-Analyse
  - Gespräch mit Kunden
  - Analyse des Entwicklungszyklus
- ISG und ASaP direkt einsetzbar
  - Konzept für Umstellung auf automatische Software-Produktion
  - ggf. Schnittstellen-Adaption
    um soviel wie möglich existierende Software übernehmen zu können
  - schrittweise Erhöhung des Automatisierungsgrades
- Umsetzung von ISG und ASaP Know-How
  - Definition eines Konzeptes für automatische Software-Produktion
  - Erstellung der benötigten Automatisierungs-Tools
    ggf. Adaption vorhandener Automatisierungs-Tools
    Ziel: soviel wie möglich existierende Software integrieren
  - schrittweise Erhöhung des Automatisierungsgrades