Softwareentwicklung in der Wissenschaft

9.12.2010

Steffen Göttsch

Gliederung

- Motivation
- Prinzipien guter Softwareentwicklung
- Praxis in der SiW
- Programmiermethoden
- Zusammenarbeit SE und Wissenschaft
- Open Source
- Green Computing
- Ausblick

Motivation

- Ihr kennt viele Prinzipien guter Softwareentwicklung
- Diese Prinzipien sind auf SE unter bestimmten Vorraussetzungen ausgerichtet

  z.B.

  - Softwarelebenszyklus von mehreren Jahren
  - Arbeit im großen Team
  - Kommerzielle Nutzung
- SiW hat oft völlig andere Gegebenheiten

Prinzipien guter SE

Ziele:

- Lesbar
- Wiederverwendbar
- Wartbar
- Erweiterbar
- Flexibel
- Einfach verwendbar
- Änderungsstabil

Prinzipien guter SE

Methoden:

- Klar strukturierter Code
- Eindeutige Bezeichner
- Kommentare
- Kapselung
- Eine Klasse – eine Aufgabe
- Tests

Auf ein bestimmtes Umfeld ausgerichtet

Praxis in der SiW

- Programm löst eine Aufgabe und wird dann entsorgt
  - kaum Wiederverwendung, Wartung, Änderung, Erweiterung
- Programmieren häufig in kleinem Team oder allein
  - Lesbarkeit weniger wichtig
- Programmierer gleichzeitig Anwender
  - Einfache Verwendbarkeit weniger wichtig

Praxis in der SiW

Gleichzeitig:

- Häufig hohe Kosten in der Rechnernutzung durch HPC
- Testbarkeit schwerer, da das Ergebnis noch nicht feststeht
- Erwartungsdruck, zwar nicht durch Kunden, aber durch Geldgeber / Förderungsprogramme
- Gewonnenes Wissen kann schnell veraltet sein – Kurze Entwicklungszeit nötig

Praxis in der SiW

Anforderungen:

- Kurze Entwicklungszeit
- Optimale Rechenzeitnutzung
- Korrektes Programmieren – try & reforge schwer durchführbar, testen nur teilweise möglich

Lebenszyklus

Klassische / kommerzielle SE:

- Lebenszyklus von mehreren Jahren
- Häufige Änderung, Wartung
- Wiederverwendung
- Verwendung durch viele verschiedene Personen

Lebenszyklus

Wissenschaftliche SE:

- Lebenszyklus von einer einzigen Ausführung
- Wiederverwendung höchstens in Teilen, häufig gar nicht
- Verwendung nur durch einen oder wenige, die gleichzeitig Entwickler sind
- Ansatz zu mehr Wiederverwendung: Open Source

Risiken der SiW

- Ergebnis steht anfangs nicht fest:
  Projekt kann komplett fehlschlagen
- Gleichzeitig hohe Datenmengen -> hohe Rechnerzeitkosten
- Verifizierbarkeit eingeschränkt

Geldgeber

Klassische SE:

- Verkaufseinnahmen (einmalig oder wiederkehrend)

SiW:

- Öffentliche Gelder
- Stiftungen

Testen

- Klassische SE: Ergebnis vorher bekannt
- Wissenschaftliche SE: Ergebnis nur vermutet oder gar nicht bekannt
- Herkömmliche Tests:
  - Ausgabe = Soll-Ergebnis: korrekt
  - Anwendungstests
- Erwartetes Ergebnis = korrektes Programm?
- Anwendungstests sehr teuer und nicht verlässlich

Programmiermethoden

Klassische „saubere“ Softwareentwicklung

- Vorher Festgelegter Entwicklungsprozess
- Klare Struktur
- Ausführlich kommentiert
- Ausführlich getestet
- Struktur auf Wiederverwendbarkeit, Änderbarkeit, Einfache Verwendbarkeit optimiert

Programmiermethoden

Vorteile:

- Sehr gut verständlich, für Arbeit in großem Team geeignet
- Wiederverwendbar, flexibel, änderungsstabil

Nachteile:

- Hohe Entwicklungsdauer
- Für viele Projekte unnötiger Aufwand

Programmiermethoden

Methode Quick & Dirty

- Kein festgelegter Entwicklungsprozess
- Simple, zielführende Programmierung für ein bestimmtes Problem
- Wenig Kommentare
- Wenig Tests
- Kein Wert auf Widerverwendbarkeit, Erweiterbarkeit, Flexibilität
- Häufig fehlende IT-Kenntnisse

Programmiermethoden

Vorteile:

- Kurze Entwicklungsdauer!

Nachteile:

- Schwer Wiederverwendbar
- Schwer nachvollziehbar und verifizierbar
- Fehleranfällig
- Schlechte Rechnerzeitausnutzung

Programmiermethoden

Angemessenes Vogehen in der SiW?

- Klarheit der Struktur und Kommentare auf die Größe des Entwicklungsteams angepasst
- Testen wo möglich und sinnvoll
- Korrektheitsprüfung per Auge
  - Teilweise Prüfung durch Tools möglich
- Auf Änderungsstabilität etc nur dann Wert legen, wenn das Programm (in Teilen) auch wiederverwendet wird

Zusammenarbeit Wiss und SE

- Entwicklung: größere Teams aus Wissenschaftlern und SElern
- SiW ↔ Klassische SE
- HPC ↔ Klassische SE
- Konferenzen mit Teilnehmern aus Wiss und SE
- Gegenseitiges Verständnis
- Bessere Zusammenarbeit

Zusammenarbeit Wiss und SE

Third international Workshop on SE for HPC Applications 26.5.2007 (SE-HPC 07)

Post-Workshop report for the Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC 07)

- Software Engeneering Forscher
- Forscher und Anwender der HPC Community
- Vorträge
- Diskussionsrunde
- Breakout Groups

Zusammenarbeit Wiss und SE

SE-HPC 07 Vorträge

- Produktivitätsmessung
- Tool Support
- IDE für SiW / HPC
- Wiederverwendung

Zusammenarbeit Wiss und SE

SE-HPC 07 Diskussionsrunde

- Research Plan vs Business Plan
- Personnel differences
- Similarities between Research Environment and Traditional Environments

Zusammenarbeit Wiss und SE

SE-HPC 07 Breakout Group HPC

- What are some software engineering techniques that have worked in the past?
- What are some things the HPC community does not need from software engineers?
- What do you most need from software engineering researchers?

Zusammenarbeit Wiss und SE

SE-HPC 07 Breakout Group SE

- What are the top things that the software engineering community has to offer the HPC community?
- What are some problems or frustrations you have had in trying to work with the HPC community or the research domain?
- What are some things that software engineers would like to offer to the HPC community, but we cannot yet?

Zusammenarbeit Wiss und SE

SE-HPC 07 Summary / Road ahead

- SE-Prinzipien hilfreich/nötig, aber speziell zugeschnitten
- Vergleich mit Bereichen mit ähnlichen Anforderungen für weitere Schritte
  - Firmeninterne Software
  - High-Risk-Software
- Besseres gegenseitiges Verständnis gewonnen

Open Source

- Wissen als öffentliches Gut
- Schnellere Entwicklung durch Wiederverwendung von vorhandenem Code
- Schnellerer wissenschaftlicher Fortschritt
- Aber auch: höhere Codeanforderungen bzgl Wiederverwendbarkeit, Verständlichkeit
- Einige Förderprogramme schreiben Open Source vor

Open Source

Open Source verfügbar:

- MIT General Curculation Model (MITgcm)
- General Estuarine Transport Model (GETM)
- General Ocean Turbulence Model (GOTM)

Green Computing

- Wissenschaft sollte den Menschen und damit auch der Umwelt dienen
- Öffentliche Gelder

- Stomsparende Geräte, Umweltschonende Produktion & Entsorgung
- Aber auch: gute Programmierung → weniger Rechenzeit !

Ausblick

Entwicklung:

- Größere, heterogene Teams
- Größere Projekte
- Wiederverwendung
- IT-Wissen wächst
- Prinzipien guter SE gewinnen an Bedeutung
- Zusammenarbeit SE / Wiss wächst
- Nach wie vor nicht überall anwendbar / sinnvoll

Viel Dank für Eure Aufmerksamkeit :)

Quellen

- Post-Workshop report for the Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC 07)
  - http://www.cse.msstate.edu/~carver/Papers/SEN_32_5.pdf
- Report from the Second International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE 09)
- Scientific Software Engineering - Basic techniques of creating practical scientific software
  - http://www.ita.uni-heidelberg.de/~pmelchior/talks/software_engineering_150410.pdf

Quellen

- Models of scientific software development
  - http://cs.ua.edu/~SECSE08/Papers/Segal.pdf
- Klassische Fehler in der Softwareentwicklung
  - http://www.theoinf.tu-ilmenau.de/~riebisch/swqs/fehler.html
- http://mitgcm.org/
- http://www.getm.eu/
- http://www.gotm.net/