Tipps

1. Kümmern Sie sich um Ihr Können

Wieso sollte man seine Zeit *mit* Software-Entwicldung verbringen, wenn man sich nicht darum kümmert, es gut zu machen.

2. Denken Sie über Ihre Arbeit nach

Schalten Sie den Autopilot ab und übernehmen das Steuer. Überdenken und kritisieren Sie Ihre eigene Arbeit ständig.

3. Bieten Sie Alternativen statt billiger Ausreden

Sagen Sie nicht, dass etwas unmöglich sei, sondern was möglich *ist*.

4. Akzeptieren Sie keine zerbrochenen Fensterscheiben

Bringen Sie schlechte Entwürfe, falsche Entscheidungen und mangelhaften Quelltext in Ordnung, wenn Sie darauf stoßen.

5. Geben Sie den Anstoß zu Veränderungen

Man kann Leute nicht zu Veränderungen zwingen. Zeigen Sie ihnen stattdessen, wie die Zukunft aussehen könnte und helfen ihnen dabei, sie mitzugestalten.

6. DenkenSie an das große Ganze

Vertiefen Sie sich nicht so sehr in Details, dass Sie nicht mehr mitbekommen, was um Sie herum passiert.

7. Machen Sie Qualität zu einer Anforderung

Beziehen Sie Ihre Anwender ein, um die wahren Qualitätskriterien für Ihr Projekt zu finden.

8. Investieren Sie regelmäßig in Ihr Wissensportfolio

Gewöhnen Sie sich an, dazuzulernen.

9. Hinterfragen Sie Gelesenes und Gehörtes kritisch

Lassen Sie sich nicht von Verkäufern, Medienrummel oder Dogmen beeinflussen. Analysieren Sie, was die Informationen für Sie und Ihr Projekt bedeuten.

10. Beides zählt: Was Sie sagen und wie Sie das tun

Großartige Ideen sind wertlos, wenn man sie nicht richtig mitteilen kann.

11. DRY - Don't Repeat Yourself (dt. Wiederholen Sie sich nicht)

Jedes Stück Wissen muss eine einzige, eindeutige und maßgebliche Repräsentation im System haben.

12. Machen Sie Wiederverwendung einfach

Wenn etwas einfach wiederzuverwenden ist, werden es die Leute auch tun. Schaffen Sie eine Umgebung, die Wiederverwendung unterstützt.

13. Beseitigen Sie Wechselwirkungen zwischen unabhängigen Dingen

Entwerfen Sie Komponenten, die jeweils einem einzigen wohl definierten Zweck dienen, die in sich abgeschlossen und unabhängig sind.

14. Es gibt keine endgültigen Entscheidungen

Keine Entscheidung ist in Stein gemeißelt. Nehmen Sie an, sie sei in den Sand geschrieben, und bereiten Sie sich auf Veränderungen vor.

15. Verwenden Sie Leuchtspurmunition, um das Ziel zu finden

Mit Leuchtspurmunition können Sie Ihr Ziel anpeilen, indem Sie Dinge ausprobieren und sehen, wie nah Sie dran sind.

16. Lernen Sie mit Prototypen

Prototypen zu bauen ist eine Lernerfahrung. Der Wert liegt nicht im Quelltext, den Sie produzieren, sondern darin, was Sie dabei lernen.

17. Programmieren Sie nahe am Problem

Entwerfen und implementieren Sie in der Sprache Ihrer Anwender.

18. Machen Sie Abschätzungen, um Überraschungen zu vermeiden

Schätzen Sie den Aufwand, bevor Sie anfangen. Damit werden Sie Probleme entdecken, bevor sie auftauchen.

19. Passen Sie den Zeitplan immer wieder mit Hilfe des Quelltextes an

Benutzen Sie die Erfahrung, die Sie bei der Implementierung sammeln, um den zeitlichen Rahmen des Projekts zu präzisieren.

20. Speichern Sie Wissen im Klartext

Klartextdateien veralten nicht. Sie helfen Ihnen, Arbeitsergebnisse länger zu nutzen und vereinfachen Fehlersuche und Testen.

21. Nutzen Sie die Macht der Kommandozeile

Benutzen Sie die Kommandozeile, wenn es grafische Benutzeroberflächen nicht bringen.

22. Verwenden Sie nur einen Editor, den aber richtig

Der Editor sollte die Verlängerung Ihrer Hand sein. Suchen Sie sich einen konfigurierbaren, erweiterbaren und programmierbaren Editor aus.

23. Verwenden Sie immer Versionskontrolle

Versionskontrolle ist die Zeitmaschine für Ihre Arbeit. Sie können *zurück* reisen.

Checklisten

✓ Neue Programmiersprachen lernen

Langweilen Sie C, C++ oder Java? Versuchen Sie es mal mit CLOS, Dylan, Eiffel, Objective C, Prolog, Smalltalk oder TOM. Jede dieser Sprachen bietet eigene Möglichkeiten und "fühlt" sich etwas anders an.. Probieren Sie eine oder mehrere davon an einem kleinen Projekt zuhause aus.

✓ Das WISDOM-Akrostichon

W hat do you want them to leam?
What is their i nterest in what you've got to say?
How sophisticated are they?
How much detail do they want?
Who owns the information?
How can you m otivate them to listen to you?

Was wollen Sie mitteilen?
Woran ist Ihr Publikum interessiert?
Wie fachkundig ist Ihr Publikum?
Wie detailliert will es Ihr Publikum wissen?
Wem wollen Sie die Information geben?
Wie motivieren Sie Ihr Publikum?

✓ Orthogonalität bewahren

- Entwerfen Sie unabhängige, wohl definierte Komponenten
- Halten Sie Ihren Quelltext entkoppelt.
- Vermeiden Sie globale Daten.
  Refaktorisieren Sie ähnliche Funktionen.

✓ Aspekte für Prototypen

Architektur
Neue Funktionalität in einem existierenden System
Aufbau oder Inhalt von externen Daten
Werkzeuge oder Komponenten von Fremdherstellern
Performance-Angelegenheiten
Benutzeroberflächen

✓ Architektur-Fragen

Sind die Verantwortlichkeiten wohl definiert?
Ist die Zusammenarbeit wohl definiert?
Ist die Kopplung minimiert?
Können Sie Quellen potentieller Verdopplung identifizieren?
Sind die Schnittstellendefinitionen und Randbedingungen akzeptabel?
Hat jedes Modell Zugriff auf die notwendigen Daten, auch dann, *wenn* es ihn braucht?

✓ Checkliste für die Fehlersuche

- Ist das erwähnte Problem die direkte Folge eines Fehlers oder bloß ein Symptom?
- Liegt der Fehler *wirklich* im Compiler? Oder im Betriebssystem? Oder doch in Ihrem eigenen Quelltext?
- Wenn Sie das Problem einem Kollegen genau erklären müssten, was würden Sie sagen?
- Wenn der verdächtige Quelltext seine Unittests besteht, sind dann die Tests vollständig genug? Was passiert, wenn Sie die Unittests mit *diesen* Daten ausführen?
- Existieren die Bedingungen, die zu diesem Fehler geführt haben, auch an anderen Stellen im System?

✓ Das Demeter-Gesetz für Funktionen

Die Methode eines Objekts soll nur folgende Methoden aufrufen:

- Methoden der aufrufenden Klasse,
- Methoden an Parametern, die übergeben wurden,
- Methoden an allen selbst erzeugten Objekten,
- Methoden an direkt enthaltenen Komponentenobjekten.

✓ Wie man überlegt programmiert

Seien Sie sich immer bewusst, was Sie gerade tun.
Programmieren Sie nicht mit verbundenen Augen.
Folgen Sie einem Plan.
Verlassen Sie sich auf verlässliche Dinge.
Dokumentieren Sie Ihre Annahmen.
Testen Sie nicht nur Ihren Quelltext, sondern auch Ihre Annahmen.
Priorisieren Sie Ihre Aufgaben.
Machen Sie sich nicht zum Sklaven der Vergangenheit.

✓ Wann man refaktorisieren sollte

- Sie haben eine Verletzung des DRY-Prinzips entdeckt.
- Sie haben etwas gefunden, was noch orthogonaler gemacht werden könnte.
- Ulr Wissen ist gewachsen.
- Die Anforderungen haben sich verändert.
- Sie müssen die Performance verbessern.

✓ Den Gordischen Knoten durchschlagen

Wenn Sie vor einem unlösbaren Problem stehen, fragen Sie sich:

- Gibt es einen einfacheren Weg?
- Löse ich das richtige Problem?
- *Wieso* ist das ein Problem?
- Warum ist es so schwierig zu lösen?
  Muss es auf diese Weise gelöst werden?
- Muss es überhaupt gelöst werden?

✓ Aspekte des Testens

Unittests
Integrationstests
Validierung und Verifikation
Ressourcenverbrauch, Fehlersituationen und Wiederherstellung (Recovery)
Performance-Tests
Usability-Tests
Die Tests selbst testen

24. Lösen Sie das Problem, nicht die Schuldfrage

Es ist egal, ob es Ihr Fehler oder der eines anderen ist.
Es ist immer noch Ihr Problem und es muss gelöst werden.

25. Keine Panik!

Halten Sie inne und *denken* Sie darüber nach, was den Fehler verursacht haben könnte.

26. "select" ist nicht kaputt

Sie werden nur sehr selten Fehler im Betriebssystem, im Compiler oder in der Software von Fremdherstellern finden.
Der Fehler liegt höchst wahrscheinlich in Ihrer Anwendung.

27. Nicht annehmen, sondern beweisen

Beweisen Sie Ihre Annahmen in der konkreten Umgebung - mit echten Daten und realen Randbedingungen.

28. Lernen Sie eine Skript-Sprache zur Textbearbeitung

Sie arbeiten einen Großteil Ihrer Zeit mit Text. Warum lassen Sie den Computer nicht einen Teil davon tun?

29. Schreiben Sie Quelltext, der Quelltext schreibt

Quelltextgeneratoren steigern Ihre Produktivität und velmeiden Wiederholungen.

30. Niemand kann vollkommene Software schreiben

Software kann nicht vollkommen sein. Schützen Sie Ihre Programme und Ihre Anwender bei unvermeidbaren Fehlern.

31. Entwerfen Sie mit Verträgen

Verwenden Sie Design by Contract, um sicherzustellen, dass der Quelltext nicht mehr und nicht weniger tut, als er vorgibt.

32. Lieber ein Ende mit Schrecken ...

Ein totes Programm richtet üblicherweise weit weniger Schaden an als ein schrottreifes.

33. Verhindern Sie das Unmögliche mit Zusicherungen

Zusicherungen bestätigen Ihre Annahmen. Nutzen Sie das, um Ihren Quelltext vor Unwägbarkeiten zu schützen.

34. Verwenden Sie Ausnahmen nur ausnahmsweise

Die Behandlung von Ausnahmen kann genauso schwer les- und wartbar werden wie klassischer Spaghetti-Quelltext. Beschränken Sie Ausnahmen auf wirklich außergewöhnliche Dinge.

35. Führen Sie zu Ende, was Sie begonnen haben

Sofern möglich, sollte die Routine oder das Objekt, das eine Ressource anfordert, auch dafür verantwortlich sein, sie wieder freizugeben.

36. Minimieren Sie die Kopplung von Modulen

Vermeiden Sie Kopplungen durch schüchternen Quelltext und mit Hilfe des Demeter-Gesetzes.

37. Konfigurieren Sie, statt zu integrieren

Realisieren Sie technologische Entscheidungen für eine Anwendung als Konfigurationsoptionen, nicht über Integration oder statische Programmierung.

38. Schreiben Sie Abstraktionen in den Quelltext, Details in die Metadaten

Programmieren Sie für den allgemeinen Fall und legen Sie die konkreten Details außerhalb des kompilierten Quelltextes ab.

39. Analysieren Sie die Arbeitsabläufe, um die Parallelität zu verbessern

Nutzen Sie Parallelisierbarkeiten im Arbeitsablauf Ihres Anwenders.

40. Entwerfen Sie Dienste

Entwerfen Sie im Sinne von *Diensten* - unabhängigen, parallel existierenden Objekten hinter wohldefinierten, konsistenten Schnittstellen.

41. Entwerfen Sie für Parallelität

Gestatten Sie Parallelität und Sie werden Schnittstellen entwerfen, die sauberer sind und weniger Annahmen treffen.

42. Trennen Sie Sicht und Modell

Gewinnen Sie viel Flexibilität fast ohne Aufwand, indem Sie in Ihren Anwendungen die Modelle von den Sichten darauf trennen.

43. Koordinieren Sie Arbeitsabläufe mit Blackboards

Verwenden Sie Blackboard-Systeme, um unterschiedlichste Fakten und Prozesse zu koordinieren. Damit bewahren Sie gleichzeitig auch deren Unabhängigkeit und strenge Trennung voneinander.

44. Programmieren Sie nicht mit dem Zufall

Verlassen Sie sich nur auf verlässliche Dinge. Hüten Sie sich vor zufalliger Komplexität und verwechseln Sie einen glücklichen Zufall nicht mit einem zielgerichteten Plan.

45. Schätzen Sie die Komplexität Ihrer Algorithmen

Entwickeln Sie ein Gefühl dafür, wie lange die Verarbeitung dauert, *bevor* Sie anfangen zu programmieren.

46. Überprüfen Sie Ihre Abschätzungen

Die mathematische Analyse von Algorithmen ist nicht alles. Messen Sie Ihren Quelltext in der ZieJumgebung.

47. Refaktorisieren Sie früh und häufig

Genauso wie man in einem Garten jätet und umpflanzt, müssen Sie Ihren Quelltext neu schreiben, überarbeiten und neu entwerfen, wenn es notwendig ist. Packen Sie das Problem an der Wurzel.

Tipps

48. Entwerfen Sie fürs Testen

Denken Sie über das Testen nach, bevor Sie anfangen zu programmieren.

49. Testen Sie Ihre Software, sonst tun es Ihre Anwender

Testen Sie schonungslos, damit Sie die Fehler vor den Anwendern finden.

50. Verwenden Sie keinen generierten Quelltext, den Sie nicht verstehen

Assistenten erzeugen Unmengen an Quelltext. Vergewissern Sie sich, dass Sie *alles* verstehen, bevor es Teil Ihres Projektes wird.

51. Nicht Anforderungen sammeln, sondern nach ihnen schürfen

Anforderungen sind selten offensichtlich. Sie sind tief vergraben unter Annahmen, falschen Vorstellungen und persönlichen Interessen.

52. Arbeiten Sie mit Anwendern zusammen, damit Sie denken wie ein Anwender

Das ist der beste Weg, um herauszubekommen, wie das System wirklich eingesetzt werden wird.

53. Abstraktionen leben länger (als Details)

Investieren Sie in die Abstraktionen, nicht in die Implementierung. Abstraktionen überleben die Flut von Anderungen durch verschiedene Implementierungen und neue Technologien.

54. Verwenden Sie ein Projektglossar

Erstellen und pflegen Sie eine einzige Quelle mit allen Fachbegriffen und Vokabeln eines Projektes.

55. Denken Sie nicht über den Tellerrand hinaus - finden Sie ihn

Wenn Sie mit unlösbaren Problemen konfrontiert sind, müssen Sie die *wahren* Randbedingungen identifizieren: "Muss das so gemacht werden? Muss es überhaupt gemacht werden?"

56. Fangen Sie an, wenn Sie bereit dafür sind

Sie haben ein Leben lang Erfahrung gesammelt. Ignorieren Sie Ihre innere Stimme nicht.

57. Einige Dinge sind einfacher getan als gesagt

Verlieren Sie sich nicht in einer Spirale von Spezifikationen. Es gibt einen Punkt, an dem man mit dem Programmieren anfangen muss.

58. Machen Sie sich nicht zum Sklaven formaler Methoden

Übernehmen Sie nicht einfach blindlings eine Technik, ohne sie zu Ihrem gewohnten Vorgehen und Ihren Fähigkeiten in Beziehung zu setzen.

59. Teure Werkzeuge machen keine besseren Entwürfe

Hüten Sie sich vor Werbetricks und Lehrmeinungen und lassen Sie sich nicht vom Preis beeindrucken. Beurteilen Sie Werkzeuge nach ihrer Leistung.

60. Formieren Sie Teams entsprechend der Funktionalität

Trennen Sie nicht Architekten von Programmierern und Tester von Datenmodellierern. Gruppieren Sie die Teams so wie die Software, die sie entwickeln.

61. Vermeiden Sie manuelle Vorgänge

Ein Shell-Skript oder eine Batch-Datei tut das Gleiche, in derselben Reihenfolge und immer wieder.

62. Testen Sie frühzeitig, häufig und automatisch

Tests, die nach jedem Build-Lauf ablaufen, sind viel effektiver als Testpläne, die im Regal verstauben.

63. Das Programmieren ist nicht getan, bis alle Tests erfolgreich waren

Das ist alles.

64. Testen Sie Ihre Tests durch Sabotage

Bauen Sie in einer separaten Kopie Ihrer Software absichtlich Fehler ein. So prüfen Sie, ob Ihre Tests die Fehler finden.

65. Testen Sie Zustandsabdeckung, nicht Quelltextabdeckung

Finden und testen Sie wichtige Zustände Ihres Programms. Nur alle Quelltextzeilen zu testen, reicht nicht aus.

66. Finden Sie Fehler nur einmal

Wenn ein menschlicher Tester einen Fehler findet, sollte es das *letzte* Mal gewesen sein, dass ein Mensch das machen muss. Wenn der Fehler wieder auftritt, soll ihn ein automatischer Test finden.

67. Behandeln Sie Deutsch wie eine Programmiersprache

Schreiben Sie Dokumente so, wie Sie ein Programm schreiben: Folgen Sie dem DRY-Prinzip, verwenden Sie Metadaten, MVC, automatische Generierung und so weiter.

68. Bauen Sie die Dokumentation ein, anstatt sie dranzuschrauben

Dokumentation, die getrennt vom Quelltext erstellt wird, ist wahrscheinlich weniger korrekt und aktuell.

69. Übertreffen Sie die Erwartungen Ihrer Anwender ein wenig

Sie müssen Verständnis für die Erwartungen Ihrer Anwender entwickeln und dann das gewisse Extra drauflegen.

70. Signieren Sie Ihre Arbeit

Früher waren Handwerker stolz, ihre Arbeiten zu signieren. Sie soJlten es auch sein.