Software-Projekte / Hilfestellung und Tipps

Inhalt

1. Einige Tipps zur Teambildung und der Arbeit im Team

Wie im allgemeinen Leitfaden erwähnt sollte wenigstens die Rolle des Projektleiters und Build-Administrators fest an ein Team-Mitglied vergeben werden.

Falls mehrere Teams gebildet werden, sollte darauf geachtet werden, dass nicht alle „Know-How-Träger“ in einem Team versammelt sind und das andere Team völlig leer ausgeht.

Die folgenden Hinweise sind aus einigen der wichtigsten bisher gemachten Erfahrungen hervorgegangen:

Zusammenhalt und Kommunikation im Team

1. Achten Sie darauf, dass Ihr Team nicht in mehrere „Lager“ zerfällt – wenn nicht alle wirklich an einem Strang ziehen, kostet das wertvolle Zeit und Nerven. Um etwas für den Zusammenhalt im Team zu tun, kann Folgendes hilfreich sein:

   a) Achten Sie auf ausreichende Kommunikation im Team! Regelmäßige Team-Meetings sind dazu ein wichtiges Mittel. Tauschen Sie evtl. auch mal Zwischenergebnisse aus: so könnten z.B. die „Architekten“ bestimmte Grundgedanken oder auch noch offene Fragen vorstellen, um den anderen Team-Mitgliedern einen Überblick vom Stand ihrer Arbeit zu geben[1].

   b) Vermeiden Sie, dass Energie in ungelösten Konflikten verloren geht: Unzufriedenheit über gewisse Umstände bzw. Entscheidungen (Arbeitsplanung, Zielsetzungen für die jeweiligen Iterationen, Art der Vorgehensweise) sollte in jedem Fall angesprochen werden – entweder in einem persönlichen Gespräch mit dem Projektleiter oder in

[1] Natürlich ergibt das keinen Sinn, solange die „Architekten“ selbst noch im Stadium völlig unausgegorener Überlegungen stecken – eine gewisse Reife muss das, was man den anderen Teammitgliedern vorträgt, schon haben, aber man muss vielleicht auch nicht warten, bis alles „fertig“ ist.

einem Team-Meeting. Es ist zwar nicht automatisch garantiert, dass sich Konflikte auflösen lassen, aber das Ansprechen eines Konflikts ist immerhin eine notwendige Voraussetzung für dessen Lösung.

c) Geben Sie sich ggf. Regeln für den Umgang miteinander: Disziplin (z.B. regelmäßiges Erscheinen zu den Teammeetings[2]), etc.

d) Wahrscheinlich wird in Ihrem Team die „allgemeine Stimmung" auch mal niedergedrückt sein – vielleicht kann Ihrem Team dann eine gemeinsame Unternehmung außerhalb der Hochschule wieder etwas auf die Sprünge helfen: einige Ihrer Vorgänger haben sich in solchen Fällen z.B. mit einem kleinen gemeinsamen Grillfest für das Team beholfen (aus den Reflexionsberichten war jedenfalls zu lesen, dass dies der allgemeinen Stimmung tatsächlich wieder aufgeholfen haben soll).

2. Manche Teams haben in der Vergangenheit versucht, möglichst nur von zu Hause aus statt gemeinsam mit Kollegen im Software-Projektlabor arbeiten – nach ein paar Monaten merkte man dann, dass der hierfür nötige Kommunikationsaufwand (trotz aller modernen technischen Kommunikationsmöglichkeiten) unterschätzt wurde und es im Projekt nicht so recht voran ging. Beim gemeinsamen Arbeiten im Labor wäre es dagegen wesentlich einfacher und effizienter gewesen, ein Problem im Software-Design oder im Quellcode unmittelbar mit einem Team-Kollegen durchzusprechen. Versuchen Sie bitte möglichst im Labor und gemeinsam zu arbeiten.

3. Achten Sie darauf, dass Team-Meetings effektiv verlaufen: Tagesordnung, Moderation, etc. Unterscheiden Sie auch, welche Fragen für die Behandlung im Plenum sinnvoll sind, und welche Fragen Detailfragen sind, die von einem kleineren Kreis geklärt werden sollten.

Zeitschätzung und Ressourcenplanung

1. Aufwandsschätzungen werden bei fehlender Erfahrung i.A. eher zu optimistisch abgegeben. Berücksichtigen Sie dies bei der Planung Ihrer Ziele für die einzelnen Iterationen.

2. Bedenken Sie, dass die zeitliche Verfügbarkeit der Teammitglieder möglicherweise nicht über das gesamte Semester hinweg gleich (gut) ist.

3. Eine stark ungleichmäßige zeitliche Belastung mancher Teilnehmer ist bereits mehrfach vorgekommen – die Abgabe Ihrer Zeitschätzungen und die Erstellung eines Arbeitsplans finden nicht nur „pro forma" statt, sondern sie sollen auch helfen, so etwas besser steuern zu können.

Analysieren und Entwerfen

1. Bei der Neu- bzw. Weiterentwicklung eines konzeptuellen Datenmodells und der Architektur kann es vorteilhaft sein, wenn sich zwei[3] oder drei Teammitglieder dieser Sache annehmen, die gut konzeptionell denken und analysieren können. I. A. ist es

[2] Das sollten Sie ohnehin als **verpflichtend** ansehen (siehe Leitfaden!)
[3] Es sollten mindestens zwei Leute sein!

ineffizient, wenn alle Teammitglieder sich vor ein leeres Blatt Papier setzen, um „eben mal schnell den Problembereich zu analysieren oder die Architektur zu entwerfen“; dabei wird oft endlos in zu viele verschiedene Richtungen gedacht und diskutiert, ohne zu einer soliden Grundlage zu kommen.

2. Diskutieren Sie alle Entwurfsentscheidungen, die Ihnen schwieriger erscheinen oder bei denen Sie sich zu unsicher fühlen, mit einem Team-Kollegen – „einsam getroffene“ Entwurfsentscheidungen ohne eine zweite Meinung bergen ein höheres Risiko von Fehlentscheidungen!

3. Denken Sie daran, dass ein konzeptuelles Datenmodell und auch eine Architektur i. A. nicht an einem Nachmittag fertig ist – die ersten Ideen sind nicht immer die besten. Ein konzeptuelles Modell und eine Architektur müssen sich entwickeln: Jeder Use-Case stellt das Modell erneut auf die Probe, häufig (besonders am Anfang) müssen dabei Ansätze verworfen und neue Ideen gesucht werden. Erst nachdem für die „architektur-relevanten“ Use-Cases ausreichend geklärt ist, dass sie mit einem bestimmten Ansatz umsetzbar sind, kann man ein gewisses Vertrauen in die Tragfähigkeit des Ansatzes haben.

4. Denken Sie beim Erstellen eines konzeptuellen Modells an den Unterschied zwischen „Was“ und „Wie“ (vgl. SWE1)

5. Der Wert eines konzeptuellen Datenmodells liegt u. A. auch darin, dass Auftraggeber und Entwicklungsteam eine gemeinsame Sprache in Form von Konzepten entwickeln können. Auch für die Kommunikation innerhalb des Teams ist das konzeptuelle Modell wertvoll. Auch UML-Diagramme, die die System-Architektur zeigen, sind wertvoll für die Kommunikation innerhalb des Teams.

Verteilung der Zuständigkeiten nach System-Komponenten – die Rolle der Architektur

1. Neben vielen anderen Vorteilen erlaubt Ihnen eine „gute“ System-Architektur eine bessere Verteilung der Zuständigkeiten im Team nach System-Komponenten: je nach Größe der Komponente gibt es einen oder mehrere dafür zuständige Entwickler, die innerhalb ihrer Komponente „weit gehend unabhängig“ arbeiten können. „Weit gehend unabhängig“ bedeutet dabei vor allem, dass es in den Komponenten eher wenig „massive“ Änderungen gibt, die über die Komponentengrenze hinausreichen und Änderungen in anderen Komponenten nach sich ziehen – die „meisten“ Änderungen bleiben lokal im Inneren der Komponenten hinter stabilen Schnittstellen verborgen. Insbesondere sollte jede Komponente möglichst hohen Zusammenhalt und möglichst schwache Kopplung zu anderen Komponenten aufweisen. Natürlich sind auch bei guten Architekturen immer wieder größere Änderungen nötig, solche Änderungen lassen sich dann aber noch halbwegs gut beherrschen und nach solchen Änderungen stellt sich jeweils wieder ein Zustand des „weit gehend unabhängigen“ Arbeitens ein – bei einer „schlechten“ Architektur ist solch ein „weit gehend unabhängiges“ Arbeiten im Extremfall kaum möglich: man steht sich aufgrund der komplexen Abhängigkeiten fast dauernd „gegenseitig im Wege“.

2. Wichtig ist, dass Änderungen an den Schnittstellen einer Komponente zu anderen Komponenten besonders sorgfältig abgestimmt werden, und dass jemand im Team die

Architektur im Auge behält, um zu verhindern, dass sie im Laufe der Zeit schleichend degeneriert.

Ein- und Auschecken - Konfigurationsmanagement

1. Arbeiten Sie beim Einchecken von Änderungen mit größter Sorgfalt – es ist bereits mehrfach vorgekommen, dass durch die Nachlässigkeit eines Teilnehmers der Rest des Teams erheblich behindert wurde.

2. Beim Auschecken werden Dateien im Allgemeinen nicht für die Bearbeitung durch Andere gesperrt; im Falle paralleler Änderungen müssen diese zusammengeführt werden („merging“). Das Zusammenführen wird im Falle von Quellcodedateien natürlich durch die Entwicklungsumgebung unterstützt. Bei manchen Dateien, die in einem proprietären Format gespeichert werden und bei denen ein Vergleich des Dateiinhalts gar nicht oder nur schlecht möglich ist (z.B. Worddokumente, UML-Diagramme, digitale Bilder, …) sollten sie für exklusive Bearbeitung sorgen, indem Sie eine Sperre („Lock“) setzen – achten Sie dann beim Einchecken auf das Entsperren.

3. Es kam schon vor, dass Teammitglieder Dateien für sich gesperrt haben, und danach wochenlang nicht mehr aufgetaucht sind – Teamkollegen, die an den Dateien ebenfalls Änderungen vornehmen wollten, waren dadurch unnötig in Ihrer Arbeit behindert. Ärgern Sie also Ihre Kollegen nicht unnötig durch solch ein Verhalten.

4. Definieren Sie vor größeren Änderungen eine (rudimentäre) Konfiguration, indem Sie im Versionsverwaltungssystem eine Bezeichnung („label“ / „tag“) vergeben, unter der Sie den alten Zustand des Systems notfalls wieder abrufen können, falls die Änderungen misslingen. Wenn Sie im Projektverlauf einen Meilenstein (i. A. am Ende einer Iteration) erreicht haben, sollten Sie auf jeden Fall eine entsprechende Bezeichnung vergeben.

2. Einige Tipps zur Vorgehensweise

Use-Case-getriebenes Vorgehen

Versuchen Sie bei der Entwicklung des Systems, ein use-case-getriebenes Vorgehen umzusetzen – zur Erinnerung nochmals einige wichtige Punkte:

Verschaffen Sie sich eine Übersicht über Akteure und Use-Cases (und andere nicht-funktionale Anforderungen). Ein (oder mehrere) Use-Case-Diagramm(e) sollten dabei das Ergebnis sein.

Priorisieren Sie Ihre Use-Cases hinsichtlich folgender Kriterien:

- Dringlichkeit für den Auftraggeber

- Was scheint einen besonders großen Einfluss auf die Architektur zu haben? („Woraus können wir am meisten über das System lernen?“)

Einigen Sie sich in Ihrem Team und mit Ihrem Auftraggeber über die Prioritäten – die im Projektauftrag genannten Anforderungen sind möglicherweise zu umfangreich für die zur

Verfügung stehende Zeit. Sie müssen also einen sinnvollen Kompromiss zwischen den Wünschen des Auftraggebers und dem tatsächlich Machbaren finden.

Nehmen Sie die Use-Cases mit der höchsten Priorität als Ausgangspunkte, um sich ins Innere des Systems hineinzudenken (jeder Use-Case gibt den Anfang eines „roten Fadens“, an dem man „entlang denken“ kann, und er stellt Architektur und Entwurf jeweils erneut auf die Probe - vgl. SWE1).

Versuchen Sie Use-Cases auch als Grobeinheiten für die Arbeitsplanung zu verwenden (jeder Use-Case benötigt ein oder mehrere Arbeitspakete zu seiner Realisierung).

Denken Sie bei der Erstellung eines konzeptuellen Datenmodells daran, dass es nicht immer rein mechanisch nach dem simplen Schema „Erst Use-Case-Beschreibungen verfassen, dann Hauptworte extrahieren …“ ablaufen kann: manchmal muss man Konzeptanalyse und Use-Case-Analyse auch miteinander verschränken.

Testen

1. Legen Sie frühzeitig fest, welche Use-Cases Kandidaten für den Abnahmetest am Semesterende sind.
2. Beginnen Sie frühzeitig mit der Entwicklung von Testfällen (lässt sich z.B. auf der Grundlage von Use-Cases bereits bewerkstelligen, falls die Use-Case-Beschreibungen ordentlich erstellt wurden).
3. Achten Sie generell auf die Testbarkeit Ihrer Anforderungsbeschreibungen.
4. Planen Sie genügend Zeit für das Testen ein.

3. Zeitliche Grobeinteilung und Termine

Siehe auch die Excel-Tabelle „Projektkalender“