



Softwarepraktikum

WS 2014/2015



- Organisation
- Thema und Anforderungen
- Ablauf
- Scrum als Vorgehensmodell
- Scrum im Softwarepraktikum



ORGANISATION



Team

- **Tutor**
Jan Hättig
- **Dozenten**
Daniel Dietsch, Marius Greitschus
- **Verantwortung**
Prof. Dr. A. Podelski



Organisation

- 6 ECTS (180h) in 14 Wochen.
- Teams mit ca. 5 Studenten (nicht B.Sc. Informatik).
- 6 Abgaben, 3 Präsentationen.
- Wöchentliches Gruppentreffen mit dem Tutor.
 - Dabei Feedback zum Projektfortschritt.
- Keine regelmäßige Vorlesung.



Organisation

- Termine
 - Betreuung
auf Anfrage Do. 14:00 – 18:00 Uhr im Pool (Geb. 82, Raum 00-029).
 - Präsentationen
 - Do. 14:00 – max. 18:00 Uhr hier.
 - Abgaben
Samstags bis 23:59 Uhr.



Dienste, Werkzeuge, Informationen

- **Wiki**
- **Informationen**
Wiki, Gruppenmitglieder, IRC, Tutoren, Poolbetreuung
- **Primäre Dienste**
SVN, Trac, Mailinglisten (sopra-crew@..., sopraXX@...), Poolrechner
- **Sekundäre Dienste**
Sonar, StatSVN, Doxygen
- **Werkzeuge**
C#, F#, .NET 4.5, XNA 4.0, Visual Studio 2010 oder neuer, ReSharper 8



Zulassung

- **Kontinuierliche Mitarbeit**
 - Belegt durch hinreichend viel **messbare** Aktivität (**SVN, Trac**).
 - **Verbrauchte Zeit** und **geschätzte Restzeit** muss im Trac angegeben werden.
 - Max. 2 Wochen nicht kontinuierlich mitarbeiten.
- **Gruppentreffen**
 - Anwesenheitspflicht.
 - 1x pro Woche 2h.
 - Max. 1x fehlen.
- **Präsentationen**
 - Anwesenheitspflicht.



Benotung

- 50% **Endprodukt**
 - Entspricht das Produkt den Anforderungen?
 - Ist das Produkt fehlerfrei (d.h. finden wir bei der Abnahme keine Fehler)?
 - Sind die Softwarequalitätsanforderungen erfüllt?
 - Wie ist die Qualität der finalen Artefakte?
- 50% **Einzelleistung**
 - Wurde die zugeteilte Arbeit *erfolgreich* erledigt?
 - Ab nächster Woche pro Woche max. 5 Punkte.
- Wenn Endprodukt oder Einzelleistung 5.0, dann Endnote 5.0.



Lernziele

- Selbstständiges Einarbeiten in unbekanntes Gebiet.
- Arbeiten im Team.
- Umgang mit Komplexität.
- Praktische Anwendung softwaretechnischer Prinzipien.



Simulation eines Softwareentwicklungsprozesses anhand eines Computerspiels.

THEMA



Sie haben die Wahl

- **Brettspiele**
 - WS 2010/2011: „Khet“
- **Space Shoot ‘em Up**
 - WS 2011/2012: „Xenon“ und „Triton“
- **Etwas eigenes**
 - Nach ihren Vorstellungen, mit uns besprochen

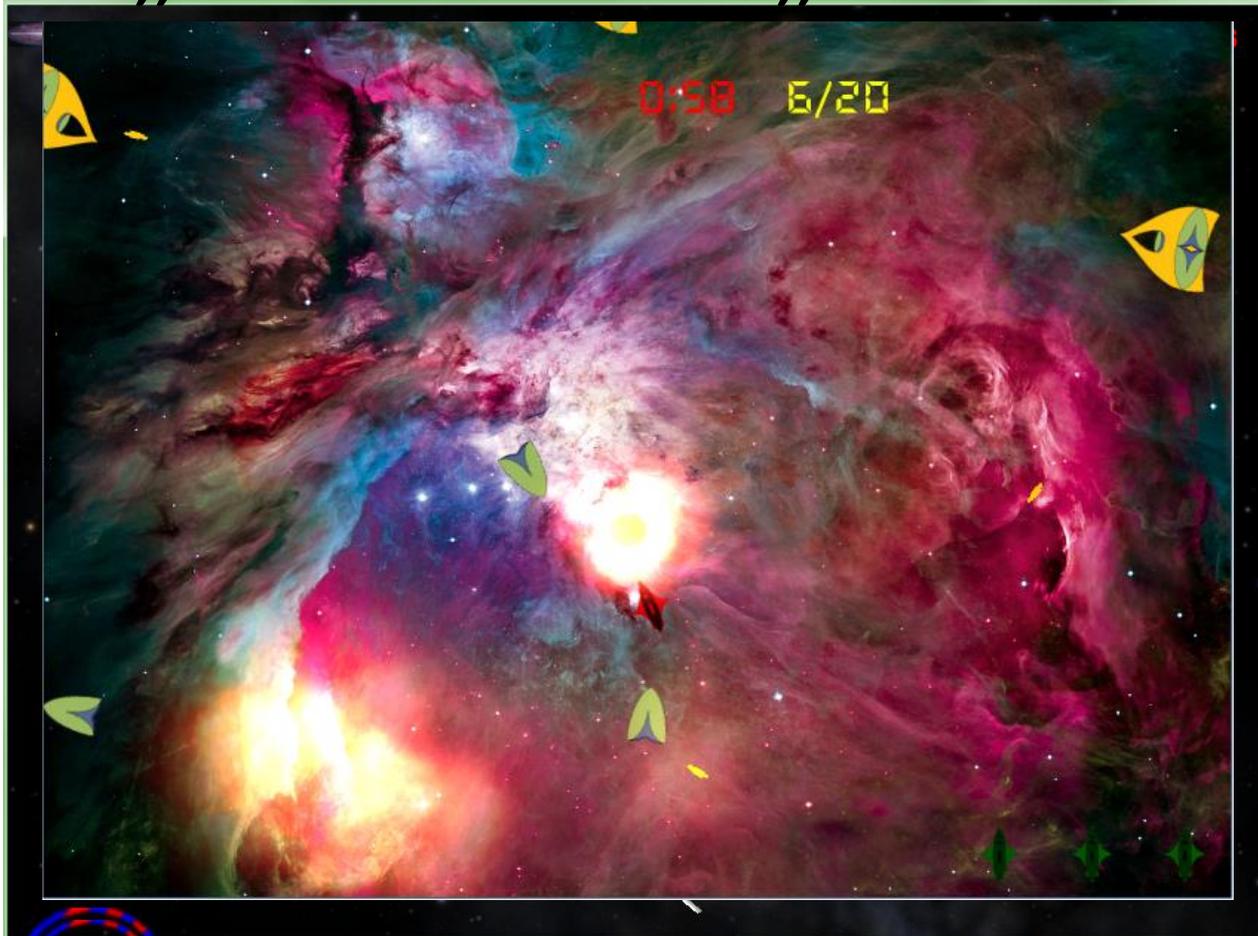


Beispiel Brettspiel: „Khet“





Beispiel Space Shoot 'em Up: „Xenon“ und „Triton“





ANFORDERUNGEN



Was sind Anforderungen?

„Anforderungen legen die qualitativen und quantitativen Eigenschaften eines Produkts aus der Sicht des Auftraggebers fest.“

Helmut Balzert. Lehrbuch der Softwaretechnik.
2. Auflage. 2000. ISBN 3-8274-0480-0



Was sind Anforderungen?

- **3 Arten** von Anforderungen
 - **Funktionale Anforderungen** definieren die funktionalen Effekte, die eine Software auf ihre Umgebung ausüben soll.
 - **Qualitätsanforderungen** beschreiben zusätzliche Eigenschaften, die diese funktionalen Effekte haben sollen.
 - **Randbedingungen** beschränken die Art, auf die die Software funktionale Anforderungen erfüllt, oder auf die die Software entwickelt wird.



Funktionale Anforderungen

- 2D oder 3D Grafik (kein ASCII).
- Potenziell zu jedem Zeitpunkt Speichern / Laden, muss aber nicht zwangsläufig vom Spieler gesteuert sein.
- Pausefunktion.
- Eigenes Menü.
- Sound.
- Alleinstellungsmerkmal.



Funktionale Anforderungen: Brettspiele

- Min. 2 Spieler, min. einer davon „menschlich“, min. ein KI-Spieler.
- Rundenbasiert oder Echtzeit.
- Komplexer als „Mensch ärgere dich nicht“.
- Exotisch.



Funktionale Anforderungen: Space Shoot 'em Up

- Min. 1 Spielobjekt wird zu jeder Zeit gesteuert.
- Echtzeit.
- Zusätzliche Mechaniken, die über einfache Bewegung und Schießen hinaus gehen.
- Mehrere Level.
- Bossgegner.



Funktionale Anforderungen: Platformer

- Min. 1 Spielfigur wird zu jeder Zeit gesteuert.
- Echtzeit.
- Direkt gesteuerte Spielfiguren müssen „springen“ können.
- Korrektes Springen muss für die Bewältigung von Leveln notwendig sein.
- Mehrere Level oder ein sehr langes Level.



Qualitätsanforderungen

- Entwickeln Sie ein **gutes** Produkt.
- Qualität der Grafik ist nicht relevant.
- Grafiken sollen in sich stimmig sein.
- Akustische Effekte sollen in sich stimmig sein.
- Richtlinien zur Bedienbarkeit von Computerspielen beachten (Wiki-Artikel „Usability beim Spieldesign“).



Randbedingungen

- Programmiersprache C# und/oder F# mit .NET 4.5.
- XNA 4.0 (XNA Game Studio 4.0 (Windows Phone SDK 7.1): 28. September 2011).
- Alternativ: MonoGame.
- Auf Windows 7 x86/x64 lauffähig.
- Visual Studio 2010 oder neuer.
- Keine Warnings oder Errors vom Compiler oder ReSharper (wöchentlich), keine Buildfehler.



ABLAUF



Woche	Organi- sation	Ent- wurf	MS 01	MS 02	MS 03	MS 04	MS 05	Was?	Wann und Wo?
0	✓	✗	✗	✗	✗	✗	✗	<ul style="list-style-type: none"> Einführungsveranstaltung besuchen Gruppeneinteilung abwarten 	<ul style="list-style-type: none"> Einführungsveranstaltung: 23.10., 14:00 - TBA, 82-00-029 Gruppeneinteilung: Im Anschluss an die Einführungsveranstaltung
1	✓	✓	✗	✗	✗	✗	✗	<ul style="list-style-type: none"> Ausarbeitung der "Zusammenfassung des Spiels" Abgabe Hausaufgabe 	<ul style="list-style-type: none"> Abgabe: 01.11. bis 23:59
2	✗	✓	✓	✗	✗	✗	✗	<ul style="list-style-type: none"> Abgabe GDD (beta) 	<ul style="list-style-type: none"> Abgabe: 08.11. bis 23:59
3	✗	✓	✓	✗	✗	✗	✗	<ul style="list-style-type: none"> Besprechung Klassendiagramm mit Tutor Präsentation des aktuellen Stands (Spielidee) 	<ul style="list-style-type: none"> Präsentation: 13.11. 15:30 - 16:00, 82-00-029
4	✗	✓	✓	✓	✗	✗	✗	<ul style="list-style-type: none"> MS01 erreicht (Spielobjekt in der Welt bewegbar, bewegliche Ansichten, Level laden/speichern, Soundausgabe) Abgabe Klassendiagramm (beta) 	<ul style="list-style-type: none"> Abgabe: 22.11. bis 23:59
5	✗	✓	✗	✓	✗	✗	✗		
6	✗	✓	✗	✓	✓	✗	✗	<ul style="list-style-type: none"> MS02 erreicht (Mehrere Spielobjekte bewegen, Interaktionen zwischen Spielobjekten, Screen-Management, Menü, HUD, Musik) 	
7	✗	✓	✗	✗	✓	✗	✗	<ul style="list-style-type: none"> Abgabe GDD (final) 	<ul style="list-style-type: none"> Abgabe: 13.12. bis 23:59
8	✗	✗	✗	✗	✓	✗	✗	<ul style="list-style-type: none"> Präsentation Programm (beta) Abgabe Programm (beta) 	<ul style="list-style-type: none"> Präsentation: 18.12. 14:00 - TBA, 82-00-029 Abgabe: 20.12. bis 23:59
9	Ferien (24.12.2014 - 06.01.2015)								
10									
11	✗	✗	✗	✗	✓	✓	✗	<ul style="list-style-type: none"> MS03 erreicht (KI bzw. Gegnerverhalten, primäre Interaktionen vorhanden, Sieg-/Niederlagebedingungen, Inhalte, Grafik/Soundeffekte) Abgabe Klassendiagramm (final) 	<ul style="list-style-type: none"> Abgabe: 10.01. bis 23:59
12	✗	✗	✗	✗	✗	✓	✓		
13	✗	✗	✗	✗	✗	✓	✓		
14	✗	✗	✗	✗	✗	✓	✓	<ul style="list-style-type: none"> MS04 erreicht (finale Version vorhanden) 	
15	✗	✗	✗	✗	✗	✗	✓		
16	✗	✗	✗	✗	✗	✗	✓	<ul style="list-style-type: none"> MS05 erreicht (Fehlerbehebung & Balancing) Präsentation Programm (final) Abgabe Programm (final) 	<ul style="list-style-type: none"> Präsentation: 12.02. 16:00 - TBA, 106-04-007 Abgabe: 14.02. bis 23:59



Hausaufgabe

- Genaue Beschreibung auf dem Wiki, grob:
 - **Werkzeuge** installieren, **Dienste** testen.
 - **Trac** kennenlernen.
 - **Texte** auf Wiki lesen
 - Clean Code, Dokumentation, Usability, Trac und SVN
 - **XNA Programm** schreiben.



Game Design Document

- GDD beschreibt die wesentlichen Merkmale des Spiels für den Auftraggeber.
 - Ähnlich zu **Lastenheft**.
- Details im Wiki.
- „Hall of Fame“.



Erster Entwurf der SW-Architektur

- Software-Architektur beschreibt **die Strukturen** eines Software-Systems durch **Bausteine** und deren **Beziehungen** und **Interaktionen** untereinander.
- Bei uns reduziert auf **Klassendiagramm** in UML.



Umsetzung

- Grob gegliedert in **5 Milestones (MS)**
 - Unsere MS beschreiben einen **Referenzablauf**.
 - Behalten Sie den Termin bei, definieren Sie sich jedoch passende Inhalte selbst.
 - Ob MS erreicht ist wird im Gruppentreffen mit dem Tutor entschieden.



SCRUM ALS VORGEHENSMODELL



Scrum

- Scrum...
 - ist ein **iteratives Vorgehensmodell**.
 - gehört zu den **agilen** Methoden.
 - sieht sich als Vertreter **empirischer** Theorien.
- Scrum besteht aus **Rollen, Events, Artefakten, Regeln**.
- Entwicklungszeit wird in **Sprints** aufgeteilt.
 - **Länge** von einer Woche bis zu einem Monat.



Rollen

- Ein Scrum-Team besteht aus
 - **Developers**
Organisieren sich selbst, verantwortlich für Qualität und Entwicklung des Produkts.
 - **Product Owner**
Sammelt und priorisiert Anforderungen, verwaltet Budget, ist verantwortlich für kommerziellen Erfolg.
 - **Scrum Master**
Löst organisatorische Probleme, ist verantwortlich für den Prozess, berät das Team.



Artefakte

- **Product Backlog**
 - Liste von **Requirements** mit abgeleiteten **User Stories**.
 - Ist geordnet nach Entwicklungsreife.
- **Requirements**
 - sind (in Scrum) sehr grob beschrieben, z.B:
 1. „Der Spieler soll Einheiten gruppieren können.“
 2. „Das Spiel soll ein Hauptmenü haben.“



Artefakte

- **User Stories**

- Beschreibung aus Sicht des Benutzers.
- Möglichst kurz.
- Nur ein Satz.
- Oft nach **Vorlage**, z.B.
 - „Als <Rolle> möchte ich <Ziel/Wunsch>, um <Nutzen>“
 - „Um <Nutzen> als <Rolle> zu erhalten, möchte ich <Ziel/Wunsch>“



Artefakte

- **Sprint Backlog**
 - Liste von **User Stories** mit abgeleiteten **Tasks**.
 - Der Aufwand einer User Story wird in **Story Points** geschätzt.
 - Für jeden Sprint wird ein neues Sprint Backlog aus dem Product Backlog abgeleitet.



Artefakte

- **Task**
 - Tasks sind Arbeitspakete, die
 - vollständig,
 - von jedem Developer bearbeitbar und
 - innerhalb eines Sprints erfüllbar sind.
 - Der Aufwand eines Tasks wird in **Personenstunden** geschätzt.



Artefakte

- **Product Increment**
 - Am Ende eines Sprints erzeugte neue Version des Produkts.
 - Sollte direkt auslieferbar sein.
- **Definition of Done**
 - Definition, die bestimmt, wann ein Task bzw. eine User-Story „fertig“ ist.
 - Wird vom Team erstellt.
 - Kann sich im Laufe des Projekts ändern.



Event: Sprint Planning Meeting

- Treffen des Teams **vor jedem Sprint**.
- Länge abhängig von Sprint-Länge (**2h pro Woche**).
- 1. **Was** wird im nächsten Sprint getan?
 - **Product Owner** präsentiert Product Backlog Einträge.
 - **Developer** wählen aus, was sie im nächsten Sprint umsetzen können.
- 2. **Wie** wird das im Sprint zu erledigende umgesetzt?
 - **Developer** zerlegen ausgewählte Einträge in kleinere Teile.
 - **Developer** erstellen neuen Software-Entwurf.



Event: Daily Scrum

- **Tägliches** Treffen.
- Begrenzt auf **15 Minuten**.
- **Developer** beantworten der Reihe nach folgende Fragen:
 - Was habe ich seit dem letzten Treffen getan?
 - Was plane ich bis zum nächsten Treffen zu tun?
 - Welche Probleme hatte ich und wo benötige ich Hilfe?
- Fragen werden **nicht** im Daily Scrum geklärt.



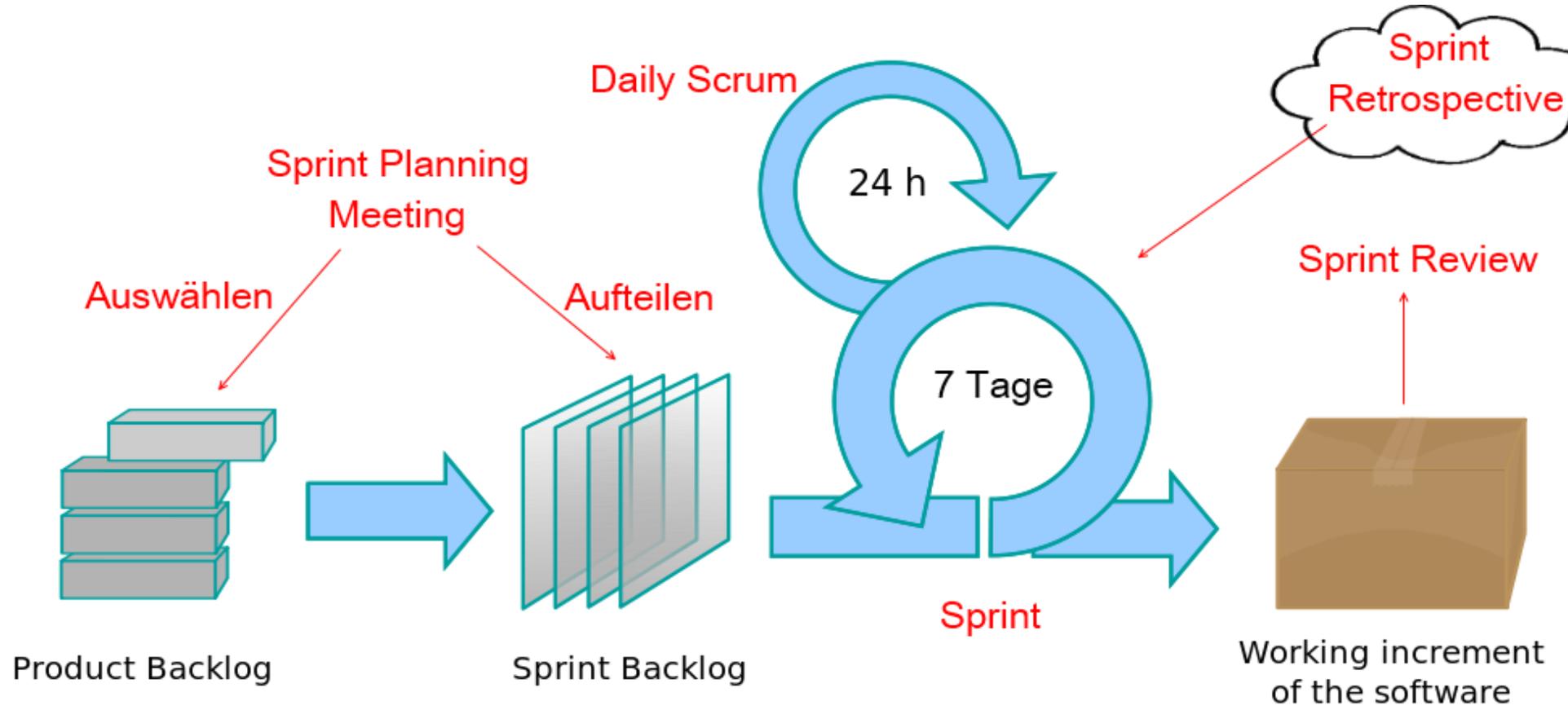
Event: Sprint Review

- Treffen des Teams **nach jedem Sprint**.
- Länge abhängig von Sprint-Länge (**1h pro Woche**).
- **Product Owner** bestimmt, welche Tasks und User Stories fertig sind.
 - Unfertige User Stories kehren in das Product Backlog zurück.
- **Developer** demonstrieren neuen Product Increment.
- **Product Owner** erklärt, wie gut die Aufwandsabschätzung war.



Event: Sprint Retrospective

- Treffen des Teams **nach dem Sprint Review**.
- Länge abhängig von Sprint-Länge (**45min/Woche**).
- **Scrum-Master** hilft, den Prozess zu verbessern:
 - Wie lief es im letzten Sprint hinsichtlich **Personen, Beziehungen, Prozessen, Werkzeugen**?
 - Ist die „**Definition of Done**“ ok?
 - Wie kann die **Arbeitsweise** des Teams verbessert werden?





SCRUM IM SOFTWAREPRAKTIKUM



Rollen

- **Tutor** ist Scrum-Master.
- **Tutor** ist Vertreter der Kunden.
- **Dozenten** sind Kunden.
- **Sie** sind Developer und Product Owner.



Artefakte im Trac

Sprint Backlog for Sprint Hausaufgabe

ID	Summary	Remaining Time	Owner	Resources	Spent Time
24	Hausaufgaben machen				
30	Student greitsch soll das Trac bedienen können, damit er in Zukunft produktiver arbeitet	7	greitsch		1
33	Scrum verstehen	5	greitsch		
34	Trac verstehen	2	greitsch		1
31	Student greitsch soll die Texte verstehen, damit er besser Spiele entwickeln kann	8	greitsch		
35	"Clean Code Development" Artikel lesen	1	greitsch		
36	"Dokumentation" Artikel lesen	2	greitsch		
37	"Usability-Prinzipien beim Spieldesign" Artikel lesen	3	greitsch		
38	"Trac und SVN" Artikel lesen	2	greitsch		
32	Student greitsch soll ein XNA Programm schreiben, damit er früh merkt, ob irgendetwas nicht funkti...	2	greitsch		
28	Programm schreiben	2	greitsch		
39	Student dzienian soll das Trac bedienen können, damit er in Zukunft produktiver arbeitet.	0	dzienian		
43	Trac verstehen	0	dzienian		
49	Scrum verstehen	0	dzienian		
40	Student dzienian soll die Texte verstehen, damit er besser Spiele entwickeln kann.	1.5	dzienian		
44	Clean Code Development' Artikel lesen	0.5	dzienian		
45	Dokumentation' Artikel lesen	0.5	dzienian		
46	Usability-Prinzipien beim Spieldesign' Artikel lesen	0	dzienian		
47	Trac und SVN' Artikel lesen	0.5	dzienian		
41	Student dzienian soll ein XNA Programm schreiben, damit er früh merkt, ob irgendetwas nicht funkti...	0.5	dzienian		
48	Programm schreiben	0.5	dzienian		
21	Totals	19			1



Event: Gruppentreffen

- Gemeinsam mit Tutor.
- Länge max. 2h.
- **Aufteilung**
 1. Sprint Review (30min)
 2. Sprint Planning (1h)
 3. Sprint Retrospective (15min)
- Verbleibende Zeit wird mitgenommen.



Sprint Review

- Neues **Product Increment** vorführen.
- Welche Aufgaben sind gemäß der **DoD** (nicht) erfüllt worden?
- **Aufwandsabschätzung** diskutieren.
- Wird der nächste **Milestone** erreicht?



Sprint Planning

- Wie viel **Zeit** hat die Gruppe im nächsten Sprint?
- Product Backlog durchgehen:
 - Das **wichtigste Requirement** für den nächsten Sprint **suchen**.
 - Requirement in User Stories und/oder Tasks **aufteilen**.
 - Benötigter Aufwand der entstandenen Items **abschätzen**.
 - **Items** in Sprint Backlog **verschieben**.
 - **Wiederholen** bis verfügbare Zeit aufgebraucht ist.
- **Items** im Sprint Backlog an Teammitglieder verteilen.



Sprint Retrospective

- Was lief im letzten Sprint **gut** oder **schlecht**?
 - Probleme mit **Teammitgliedern**.
 - Probleme mit dem **Prozess**.
 - Probleme mit der **Zeiteinteilung**.
 - Probleme mit **Tools**.
- **Schriftlichen Plan** machen, was verändert werden soll.
- Bei Bedarf die **Definition of Done** anpassen.



Hinweise

- **Abhängigkeiten** zwischen den Aufgaben berücksichtigen.
- Wenn Aufgaben nicht geschafft werden:
 - **Rechtzeitig** an die **Gruppenliste** kommunizieren (Punkte).
- DoD bestimmt Ihre Einzelnote.
 - **Minimalanforderung**: Ticket geschlossen und durch Tutor „abgenommen“.
 - DoD steuert Qualität.



Hinweise

- **Gemeinsam arbeiten**
 - Termine finden, an denen man gemeinsam arbeiten kann (auch von zu Hause aus via Skype, IM, IRC, ...).
 - Kurzes „Daily Scrum“ am Anfang solcher Treffen.
- **Aufwandsabschätzung** ernst nehmen.
- Organisation in **wiederkehrenden Aufgaben** strukturieren.
- Alle Probleme **früh** ansprechen.
 - In der Gruppe.
 - Direkt mit dem Tutor.
 - Direkt mit den Dozenten.



Was nun?

1. Gruppeneinteilung.
2. Pool Account besorgen?
3. E-Mail Adressen für Gruppenliste austauschen.
4. Regelmäßigen Termin für das Gruppentreffen ausmachen.

Ab morgen:

- Hausaufgabe machen.
- Treffen Sie sich mit Ihrer Gruppe und dem Tutor.
- Entwickeln Sie eine Spielidee.
- Machen Sie sich mit den Werkzeugen und Techniken vertraut.
- Legen Sie Ihre erste Definition of Done im Gruppentreffen fest.



FRAGEN?