Projektmanagement

“Braucht es dieses Feature wirklich?”, Arbeitszeiterfassung, (dezentrale) Zusammenarbeit

- Link collection project management
- Erfahrungen aus den Projekten – Projektmanagement Umfrage
- „Ich bin ja kein Entwickler“
- Zeigt her, woran ihr arbeitet!

□□ Link collection project management

To read, listen & watch

- *„Brauchst du dieses Feature wirklich?“ und 11 weitere Learnings*: https://prototypefund.de/12-learnings/

Methods & tools

Methods & frameworks

- *User Story Mapping* is useful to plan which features to work on first. It can be useful alone, but it is particularly helpful for working in teams: https://jpattonassociates.com/the-new-backlog/
  - There is also a Thema, *User Story Mapping* by Jeff Patton: https://www.oreilly.com/library/view/user-story-mapping/9781663728661/
- *Kanban* is a relatively lightweight project management method. Think of it as a slightly expanded todo list: https://en.wikipedia.org/wiki/Kanban
- *Stand-up-meetings* usually happen daily: Meet with the team, everyone tells everyone else the answers to these 3 questions: What did I do since the last stand-up? What do I want to do today? What is blocking me? Keep these short. It is meant as a mutual short update. Don't go into discussions, note questions down and ask them after the stand-up: https://en.wikipedia.org/wiki/Stand-up_meeting#Software_development
- *Governance Readiness Checklist*: https://sustainoss.org/governance-readiness/

Tools

- There are various open source project management tools based on the idea of having virtual cards or sticky notes. If you already use an issue tracker or git hosting, a similar feature might be already implemented there.
  - https://wekan.github.io/
  - https://taiga.io/
  - https://kanboard.org/
- Whiteboards and FOSS-alternatives for Miro:
  - https://excalidraw.com/

- https://www.tldraw.com/

□□ Erfahrungen aus den Projekten – Projektmanagement Umfrage

Wir haben eine Mini-Umfrage veranstaltet und die Teams aus vergangenen Runden nach ihrer Erfahrung im Projektmanagement gefragt. Hier seht ihr, welche Tools andere Teams genutzt haben und was sie daraus gelernt haben. Wenn ihr auch eure Erfahrungen teilen möchtet, schreibt eine E-Mail an info@prototypefund.de.

Alli - Begleitung bei der medikamentösen Schwangerschaftsbeendigung

Wie groß war euer Team?

4

Wie habt ihr die Aufgaben verteilt?

Wir haben ein Orga-Team aus 2 Personen, und innerhalb dessen Verantwortungen verteilt und transparent kommuniziert - Das funktioniert für uns sehr gut!

Welche Werkzeuge?

Slack, Notion, Asana, Documents

Was würdet ihr anders machen?

Wir würden von Anfang an ein Orga-Team haben. Vorher haben wir versucht die Aufgaben gleich zu verteilen, das hat aber nicht gut funktioniert und mehr Stress und Frust als Nutzen gebracht.

Außerdem haben wir mittlerweile eine Wochen-Übersicht darüber, wer aus dem Team wann wie kann. Es hat sich herausgestellt, dass es nur ganz wenige Zeiten gibt, an denen das ganze Team kann. Die Terminplanung ist jetzt viel einfacher, das würden wir das nächste mal auch von Anfang an so machen.

Leicht bedienbares Konfigurationstool für QUBES OS

Wie groß war dein Team?

1

Wie hast du die Aufgaben verteilt?

Ich habe alles gemacht □□

Welche Werkzeuge hast du benutzt?

eine todo.txt Datei (http://todotxt.org/) mit klar definierten Prioritäten, ein Whiteboard mit Haftnotizen

Was würdest du anders machen?

Nicht allein arbeiten, nicht krank sein? Im Großen und Ganzen bin ich mit meiner Projektmanagement zufrieden. Auch wenn ich Gesundheitsprobleme hatte, habe ich alles gut genug geplant um das Projekt pünktlich fertigzustellen.

Tuttle

Wie groß war euer Team?

4

Wie habt ihr die Aufgaben im Team verteilt?

Projektmanagement über Github

Welche Werkzeuge habt ihr verwendet?

boxtribute

Wie groß war euer Team?

4

Wie habt ihr die Aufgaben im Team verteilt?

1 Produktmanagerin, 1 Backend-Entwickler und Projektleiter, 1 Frontend-Entwickler, 1 Fullstack-Entwickler

Welche Werkzeuge habt ihr verwendet?

Trello, Slack, Github (Pull requests)

„Ich bin ja kein Entwickler“

*Nicht nur Softwareentwickler*innen entwickeln Software! Was es in einem FOSS-Projekt sonst noch so zu tun gibt, erklärt Meike vom Project Accidental Contributions (Runde 16).*

Es ist rund 20 Jahre her, dass ich hinter den Vorhang Freier Software blickte. Das klang (und klingt heute noch) sehr verheißungsvoll: alle können mitentwickeln, können Verbesserungen und neue Funktionen einbauen oder Fehler beheben. Dem ersten Impuls „Ich bin dabei!“ folgte unmittelbar der zweite: „Ach nee, ich kann ja gar nicht programmieren.“ Auch das dürfte vielen heute nicht anders gehen.

Softwareentwicklung und Programmieren werden allzu oft als Synonyme gedacht, ein weit verbreiteter Irrglaube. Wie schön wäre es, wenn sich künftig ein dritter Impuls anschließen würde: „Aber ich habe andere Fähigkeiten; wo soll ich anfangen?“

Das Full-Stack-Multitalent – ein Modell von gestern

Ich bin noch in einer Zeit IT-sozialisiert worden, in der man von „Homepages programmieren“ sprach und weitgehend reine HTML-Projekte meinte. So als würde man „programmieren“, wenn man ein Textdokument strukturiert. Und es klang, als handele es sich dabei um Geheimwissen, für das man jahrelang studiert haben musste und nicht um etwas, das man sich an zwei Wochenenden im Selbststudium draufschaffen konnte. Meine zwei Wochenenden hatten mich also zur Geheimwissenden gemacht, und so konnte ich gut mein Studium finanzieren.

Daran muss ich heute denken, wenn ich die Ehrfurcht vor den Menschen mit Programmierkenntnissen höre. Nicht falsch verstehen, ich habe auch Respekt vor Programmierer*innen. Es ist ein Job, in dem man sich nicht auf Gelerntem ausruhen kann. Aber es sind eben auch Fähigkeiten, die dank freiem Wissen im Selbststudium erlernbar sind. (Gut, man braucht auch mehr als zwei Wochenenden.)

Diese Ehrfurcht schafft eine Kluft, die Menschen mit anderen Talenten davon abhalten mag, Softwareentwicklung als etwas zu betrachten, in das sie sich einbringen können. Deshalb schreibe ich diesen Blogpost: als Aufforderung zu mehr Selbstbewusstsein.

Natürlich gibt es Multitalente. Viele Programmierer*innen mussten zwangsläufig verschiedenste Gewerke beherrschen, schlicht weil es mangels besseren Wissens von ihnen erwartet wurde. In Stellenanzeigen finden wir immer noch vielfach nicht näher spezifizierte „Full-Stack-Entwickler“-Gesuche. Was auch immer man sich heute darunter vorstellen mag (sämtliche

Programmiersprachen, viele Frontend-Frameworks, alle Gewerke?)

Softwareentwicklung braucht viele Talente

Dröseln wir doch einmal auf, welche Rollen ein typisches Softwareentwicklungsprojekt benötigt. Ich konzentriere mich hier auf Open-Source-Projekte, die oftmals dezentral und asynchron entwickelt werden. Diese Liste ist selbstverständlich nicht abschließend und Rollen können auch von mehreren Personen (oder mehrere Rollen von einer Person) übernommen werden.

Konzeption

Konzepter*innen führen Bedarfsanalysen durch, übersetzen Ziele und Wünsche in lösbare Anforderungen, zeigen Widersprüche oder Lücken in der Vorplanung auf. Sie erstellen in Abstimmung mit weiteren Rollen ein Konzept. Hilfreich sind Talente in folgenden Bereichen:

- Empathie und Kommunikation
- Folgenabschätzung
- Texte schreiben
- Grundverständnis von Softwarebedienung
- Grundverständnis von Programmlogik

UX-Design

Nein, nicht „Design“. Hier geht es um die Gestaltung von Prozessen. UX steht für User Experience, zu deutsch: Benutzungserfahrung. Beabsichtigt ist eine *positive* Erfahrung. Dieses Gewerk hat einige Nähe zur Konzeption, ist aber typischerweise auch für die Visualisierung zuständig.

Als künftige Nutzende werden exemplarische „Personas“ entwickelt, deren Nutzungsszenarien die Entwicklung begleiten sollen. Für eine Anwendung mit grafischer Oberfläche werden beispielsweise optimale Klickpfade erarbeitet und der Ablauf in Form von Wireframes (Skizzen) grafisch dargestellt. Hilfreiche Fähigkeiten:

- Grundverständnis für Psychologie
- Empathie und Kommunikation
- Kenntnis unterschiedlicher Software-Bedienkonzepte (Erwartungshaltung basierend auf Erfahrungen mit vergleichbarer Software)
- Wissen über Barrierefreiheit
- Erfahrung mit Usability-Methodik
- Erfahrung mit Visualisierungssoftware
- Texte/Beschriftungen für User-Interaktionen schreiben

Programmieren

Geschenkt. Softwareentwicklung kommt nur in ganz seltenen Fällen ohne Programmieren aus. In diesem Blogpost soll es ja darum gehen, welche Gewerke außerdem gebraucht werden.

Design

Ja, jetzt meine ich tatsächlich die visuelle Gestaltung. Nebenbei bemerkt, das Deutsche versteht unter Design sehr eng das Sichtbare, wohingegen das Englische hier einen sehr weiten Begriff von Gestaltung hat, der auch Planung und Konstruktion umfasst.

Nicht jedes Softwareprojekt braucht eine grafische Oberfläche, es gibt auch durchaus viele Programme, die „in der Konsole“ bzw. einem Terminal-Fenster ausgeführt werden, also prinzipiell textbasiert. Doch auch hier finden zunehmend Visualisierungen statt, beispielsweise durch farbige Hervorhebungen von Warnhinweisen oder Fortschrittsbalken. Auch hier können Designer:innen wertvolle Beiträge leisten.

Leider wird dieses Gewerk oft als „Hübsch-machen“ missverstanden. Design sorgt aber für Verständlichkeit und Orientierung, für erlernbare Bedienprinzipien und Schutz vor Überfrachtung. Farbgebung und Illustrationen geben einem Projekt Wiedererkennbarkeit und früher oder später stellt man fest, dass man für nahezu jedes Tool ein Logo (nebst Logovarianten) haben möchte. Und sei es nur, damit das Projekt auf GitHub nicht so nackig aussieht.

Hilfreich neben der Design-Erfahrung:

- Erfahrung mit responsivem Design (es gibt keine „normale“ Bildschirmbreite)
- Wissen über Barrierefreiheit
- Teamfähigkeit und Kommunikation

Texting

Wir haben sicherlich alle schon mal über unverständlichen Fehlermeldungen gegrübelt. Man könnte daraus schließen, dass die Multitalent-Entwickler*innen dort kein voll besetztes Team hatten. Die Fähigkeit, zielgruppengerechte Texte zu schreiben, ist wohl die am meisten unterschätzte im Softwareentwicklungsbereich.

Texter*innen kümmern sich um knappe und zugleich verständliche Textinformationen unter Verwendung zielgruppengerechter Formulierungen. Beispielsweise die Beschriftung und Platzhaltertexte von Formularelementen, unmissverständliche Handlungsaufforderungen oder Rückfragen („Alle Elemente nicht löschen? – Ja|Nein“) oder unterstützende Hilfetexte ohne befremdlichen Fachjargon. Typischerweise gehört auch das Übersetzen englischer Textelemente des Quellcodes in die Oberflächensprache dazu. Hilfreich sind hier also:

- Englischkenntnisse
- Kenntnis der Sprachvoraussetzungen der Zielgruppe

- Fähigkeit, sich auf unterschiedliche Sprachstile einzustellen (vorzugsweise einschließlich einfacher Sprache)
- Bereitschaft, Texte in Programmcode zu suchen

Projektmanagement

Dieses Gewerk ist gerade im Bereich der Freien Software stark unterschätzt. Natürlich gibt es die erfolgreich sich selbst organisierenden Teams. Aber unzählige Vorhaben drohen unterzugehen, wenn sich niemand für die Fertigstellung zuständig fühlt.

Sei es die Definition von „fertig“ oder zumindest „gut genug für einen ersten Wurf“, bei der eine zweite Perspektive in einem 1-Personen-Team schon ungemein hilfreich ist. Oder sei es die Koordinierung der verschiedenen Gewerke, das Klären von Verfügbarkeiten, wenn man auf jemandes Zuarbeit wartet. Oder das Organisieren regelmäßiger Status-Update-Termine. Projektmanagement ist schon allein deswegen hilfreich, weil es mindestens einem weiteren Menschen nicht egal ist, ob oder wann ein Projekt fertiggestellt wird.

Durchgängiges Projektmanagement sorgt außerdem dafür, dass Informationen an einer Stelle zusammenlaufen und dokumentiert werden. Das verhindert, dass Teilprojekte von neuen Beitragswilligen immer wieder neu formuliert und angegangen werden. (Ihr glaubt gar nicht, wie oft das im ehrenamtlichen Bereich passiert.) Was du dafür brauchst:

- Empathie und Kommunikation
- Organisiertheit
- Freude an Dokumentation
- Motivationstalent
- Geduld

Testen/Qualitätssicherung/Fehler melden

Softwareentwicklung profitiert von fortlaufenden Tests am meisten, wenn der jeweilige Arbeitsstand von Menschen getestet wird, die den Code nicht selbst geschrieben haben. Gemeint sind hier nicht Usability-Tests mit vollkommen Fremden (darum kümmert sich eher das Gewerk UX-Design). Das geht auch ganz niedrigschwellig: „Kannst du dir das mal rasch anschauen?“

Ob Code-Qualität oder grafische Bedienelemente, Fehlermeldung oder die Prüfung auf Barrierefreiheit: es findet sich bestimmt jemand, den man fragen kann. Für diesen Blogpost belasse ich es dabei, dass sich jede:r ungeachtet der eigenen Talente eingeladen fühlen darf, Software zu testen und Rückmeldung zu geben. Allenfalls erforderlich:

- Respektvolle Kommunikation

Öffentlichkeitsarbeit

Sinnstiftend ist die entwickelte Software erst, wenn sie auch genutzt wird. Und dazu muss erst einmal bekannt gemacht werden, dass es sie gibt. Marketing, PR (public relations) oder eben Öffentlichkeitsarbeit sind die Gewerke, die sich darum kümmern.

Dafür sollte erst einmal eine Strategie erarbeitet werden: Wer nochmal sind unsere Zielgruppen? Wo erreichen wir sie und wofür brauchen sie die Software? Wozu wollen wir sie auffordern und wie messen wir den Erfolg der Maßnahmen?

Je nach Zielgruppe wird dort darüber gesprochen, wo sich Interessierte finden können. Das können Vorträge auf Open-Source-Konferenzen sein (idealerweise mit Aufzeichnung), Interviews bei einschlägigen Podcasts, Pressemitteilungen, Informationen auf Social-Media-Plattformen … oder auch bezahlte Werbeplätze. Aber für letztere ist bei Open-Source-Software nur in den seltensten Fällen Budget vorhanden. Besonders hilfreich ist hier:

- Zielgruppen-/Branchenkenntnis
- Kommunikations- und Begeisterungstalent
- Erfahrung mit Open-Source-Communities („Marktschreierei“ und haltlose Produktversprechen vermeiden)
- Organisationstalent

Softwareentwicklung ist Teamarbeit

Auch wenn viele von uns mehrere dieser Gewerke beherrschen, halte ich es für wichtig, dass wir uns zumindest klarmachen, dass das so ist. Damit wir uns fragen, in welchem Job wir gut sind oder welchen wir gern machen und wobei wir uns besser Unterstützung holen sollten.

Es ist okay, wenn du als Programmiererin keine gute Texterin bist. Solange du es dir bewusst machst und die Aufgaben zusammenschreibst, damit jemand anderes den Job machen kann. Es ist okay, wenn du als Projektmanager auch für Konzeption und Qualitätssicherung verantwortlich bist. Du darfst aber die gute Idee kundtun, einen Sparringspartner zu benötigen.

Und du, die du vorhin noch den Impuls hattest, „Ich kann ja gar nicht programmieren“, du bist jetzt hoffentlich motiviert, deine Fähigkeiten einzubringen. Programmierer*innen haben wir in der Open-Source-Landschaft schon genug. Wir brauchen aber dringend deine Fähigkeiten!

„Ich bin dabei – wo soll ich anfangen?“

Du kennst bestimmt ein Open-Source-Projekt, das dir persönlich wichtig ist oder bei dem dich schon immer eine Kleinigkeit gestört hat. Suche nach einer offiziellen Website oder einem entsprechenden Projekt auf GitHub. Suche zunächst nach Contribution guidelines. Üblicherweise musst du dich zuerst registrieren. Wenn du genau hinschaust, wirst du nirgendwo lesen, dass du für deine Beiträge bestimmte Qualifikationen mitbringen musst ;-)

Mach dir bewusst, welche Fähigkeiten du anbieten und einbringen kannst und stelle dich damit vor. Je nach Community könnte die Kommunikation z. B. über eine Mailingliste oder ein Chatsystem stattfinden, das findest du schon. Lege etwaige Selbstzweifel ab. Allein deine Bereitschaft, etwas beizutragen, ist schon ein Grund zur Freude. Du kannst dich natürlich auch schon auf die Suche begeben, wo du deinen Verbesserungsvorschlag vorstellen kannst oder wo eine Aufgabe beschrieben ist, die du dir zutraust. Und sei nicht frustriert, wenn eine Reaktion nicht sofort erfolgt. So arbeiten wir halt: dezentral und zeitversetzt.

Und wenn du selbst dich in keinem der Gewerke siehst, kannst du zumindest schon etwas beitragen, indem du diesen Artikel mit anderen teilst, das wäre ja auch schon Öffentlichkeitsarbeit.

Wirklich alle können etwas zur Open-Source-Entwicklung beitragen!

□□ Zeigt her, woran ihr arbeitet!

*Beim Prototype Fund erhalten viele Entwickler*innen erstmals die Möglichkeit, sich sechs Monate lang auf ihr Herzensprojekt zu konzentrieren. Was richtig produktiv sein kann, kann auch zu neuen Herausforderungen führen: Wie koordiniert man eigentlich die eigene Arbeit, wenn einem keiner sagt, was man zu tun hat? Im Gastbeitrag stellt Katharina Rasch vom Projekt Trans-Europa-Planer einen Vorschlag vor, um mit der eigenen Zeit zu haushalten.*

Ihr hattet eine tolle Projektidee und habt die Zusage für den Prototype Fund bekommen. Ihr freut euch riesig und stürzt euch direkt in die Arbeit: Sechs Monate Zeit, los geht's!
Zu Beginn kommt ihr richtig schnell voran, ihr freut euch, bald eine erste Version mit euren Nutzer*innen ausprobieren zu können. Nur noch kurz Feature X, Y, Z fertigstellen, noch ein bisschen am Design feilen, und und und… Und dann, dann hakts, Feature Y ist viel schwieriger als erwartet, ihr kommt nicht weiter, und plötzlich sind 5 Monate um und mit einem anderen wichtigen Teil des Projektes habt ihr noch gar nicht angefangen.

Schluss damit! Wenn ihr euer Projekt niemandem zeigt, könnt ihr nicht wissen, ob ihr auf dem richtigen Weg seid, besteht immer das Risiko, sich im Kleinklein zu verfangen, in irgendeinem Detail zu vergraben und das große Ganze aus dem Blick zu verlieren.

Deswegen: zeigt her, woran ihr arbeitet! Und zwar regelmäßig. Regelmäßig, das heißt jeden Freitag, oder jeden zweiten Montag oder wie es für euch am besten passt. Und, ganz wichtig: Genau an diesem Tag legt ihr auch immer fest, was ihr beim nächsten Mal zeigen wollt. Damit erreicht ihr gleich zwei Ziele: Erstens fokussiert ihr euch mehr. In einem klar abgegrenzten Zeitraum konzentriert ihr euch auf einen bestimmten Aspekt eures Projektes. Zweitens schafft ihr euch regelmäßig den Raum einen Blick aufs große Ganze zu werfen und zu entscheiden, welcher Aspekt des Projektes als nächstes am Wichtigsten ist.

Die Strategie

Schritt 1:

Legt fest, was für euch *regelmäßig* bedeutet. Für mich als Einzelprojekt in Vollzeit hat sich „jeder Freitag" bewährt. Wenn ihr in Teilzeit oder in einem kleinen Team an eurem Projekt arbeitet, ist aber vielleicht ein Termin alle zwei Wochen sinnvoller. Größere Abstände würde ich nicht empfehlen, denn je länger der Zeitraum ist, desto weniger scharf bleibt der Fokus und desto seltener die Chance den Fokus zu ändern.

Schritt 2:

Sucht euch eine Person, die sich für euer Projekt interessiert und der ihr regelmäßig euren Arbeitsfortschritt zeigen könnt und besprechen könnt, was als nächstes ansteht. Die Person braucht (in den meisten Fällen) keinen technischen Hintergrund, aber darf gern auch mal kritische Fragen stellen. Es wäre gut, wenn die Person pro Woche bzw. alle zwei Wochen circa eine halbe Stunde Zeit für euch hat. Wenn nötig können sich auch mehrere Personen abwechseln. Bei Team-Projekten kann die Person auch ein Teammitglied sein, welches nicht so stark in die alltäglichen Aufgaben eingebunden ist, besser ist aber immer jemand externes.

Schritt 3:

Regelmäßig zeigt ihr dieser Person, was ihr seit dem letzten Mal geschafft habt. Und dann plant ihr gemeinsam, was ihr nächstes Mal zeigen wollt. Am besten habt ihr dafür bereits einen Vorschlag, aber seid auch bereit zuzuhören, wenn die Person findet, dass Anderes vielleicht aktuell eine höhere Priorität hat.

Es ist wichtig, dass das Ziel klar formuliert ist. Ein paar Beispiele wären:

- Wir stellen 3 Beispielreisen (Berlin nach Prag/London/Venedig) auf einer Karte dar.
- Wir führen 5 User Tests durch und stellen die Resultate strukturiert vor.
- Wir stellen eine Landing page für das Projekt unter einer eigenen Domain online. Folgende Punkte sollen auf der Page abgedeckt sein: A, B, C.

Schritt 4:

Ihr setzt euer Ziel um. Und klar, da geht manchmal auch was schief und was ihr machen wolltet, ist in den ein/zwei Wochen doch nicht erreichbar. In dem Fall ist es am besten, wenn ihr versucht, schwierige Details wegzulassen, um trotzdem das grobe Ziel erreichen zu können. Beispielsweise könntet ihr erstmal ein paar Testdaten hardcoden, statt sie, wie geplant, aus einer Datenquelle zu scrapen. Oder die Landing Page hat dann doch erstmal nur Überschriften, aber noch keine Inhalte. Und wenn das immer noch nicht ausreicht, dann ist das auch nicht so schlimm, dann habt ihr zumindest gelernt, welche Schwierigkeiten es gibt und könnt bei der nächsten Planung besser einschätzen, was realistisch erreichbar ist.

Warum funktioniert das?

Für mich funktioniert diese Strategie aus mehreren Gründen:

- Ich will gern schaffen, was ich versprochen habe und es motiviert mich total, jede Woche etwas zeigen zu können.
- Ich habe einen klaren Fokus für die Woche. Insbesondere habe ich an Montagen weniger Schwierigkeiten, zurück ins Projekt zu kommen.

- Falls ich mich doch mal tief in ein Detail vergrabe, werde ich spätestens am Freitag da rausgeholt und arbeite dann in der nächsten Woche lieber erstmal an etwas anderem.
- Während des gesamten Projektzeitraums behalte ich das große Ganze im Blick. Und die externen Augen helfen mir, wenn nötig meinen Kurs zu korrigieren.

Und wem das jetzt alles sehr bekannt vorkam: Ja, die Strategie ist natürlich stark an Methoden aus der agilen Software-Entwicklung angelehnt, angepasst auf kleine Teams bzw. Einzelpersonen und mit extrem wenig Overhead. Ihr arbeitet in ein- oder zweiwöchigen Sprints, am Ende jedes Sprints gibt es eine Demo und die externe Person spielt euren Product Owner.

Probiert’s doch mal aus und beobachtet, was für euch funktioniert und was nicht, und passt die Strategie entsprechend für euch an. Wichtig bleibt nur: **Zeigt her, woran ihr arbeitet!**

Katharina Rasch arbeitet freiberuflich als Data Scientist und Software Developer in Berlin. In der 16. Runde des Prototype Fund entwickelte Katharina den Trans-Europa-Planer, um komplizierte Bahnreisen quer durch Europa einfacher planbar zu machen.