Dokumentacja do oprogramowania

poradnik dla managerów

Paweł Kowaluk

przedsłowie: Michał Skowron

Przedsłowie

Bardzo się ucieszyłem kiedy Paweł powiedział mi, że pisze książkę o dokumentacji do oprogramowania. Zanim nawet dowiedziałem się konkretnie o czym była. Od razu wiedziałem, że był to dobry pomysł. Po pierwsze, dlatego, że literatury branżowej w naszym ojczystym języku jest jak na lekarstwo. Po drugie, dlatego, że Paweł jest jedną z garstki znanych mi osób, które mogłyby, a wręcz powinny, napisać taką właśnie książkę. Jego wieloletnie i bogate doświadczenie w komunikacji technicznej, nietuzinkowy zestaw umiejętności oraz unikalne spojrzenie na tę branżę sprawiają, że ta niewielka książka ma w sobie duży ładunek wiedzy. Oczywiście, można by napisać kilka tomów na temat dokumentacji do oprogramowania, ale Paweł wyłuskał dokładnie to co chciałbym wiedzieć jako osoba kompleksowo zajmująca się dokumentacją w organizacji.

Kiedy na początku 2012 rozpoczynałem swoją przygodę z komunikacją techniczną, mogłem tylko pomarzyć o takiej publikacji w języku polskim. Moje życie na pewno byłoby wtedy łatwiejsze, bo na początku trafiłem do firmy gdzie byłem pierwszym i jedynym Technical Writerem. Jak się łatwo domyślić, musiałem od zera i kompleksowo zająć się dokumentacją do oprogramowania. Czyli potrzebowałem dokładnie takiej wiedzy jaka zawarta jest w tej książce.

Na koniec dodam, że moja radość z tej publikacji jest tym większa, że Paweł zdecydował się ją udostępnić w ramach darmowej wymiany wiedzy. Ale to w sumie trochę gorzej dla Ciebie, bo teraz już ciężko Ci będzie znaleźć wymówkę, żeby jej nie przeczytać - jest dobra,

krótka i za darmo. Chyba nie dało się tego lepiej zrobić.

Michał Skowron

Wprowadzenie

Dla kogo jest ta książka?

Ta książka została napisana z myślą o osobach odpowiedzialnych za dokumentację w firmie. Nieważne czy jesteś na stanowisku kierowniczym i prowadzisz zespół, czy jesteś product ownerem i w Twoim produkcie nie ma jeszcze dokumentacji czy jesteś jedynym technical writerem w firmie. Jeżeli Twoim zadaniem jest kompleksowo zająć się dokumentacją, to ta książka jest dla Ciebie.

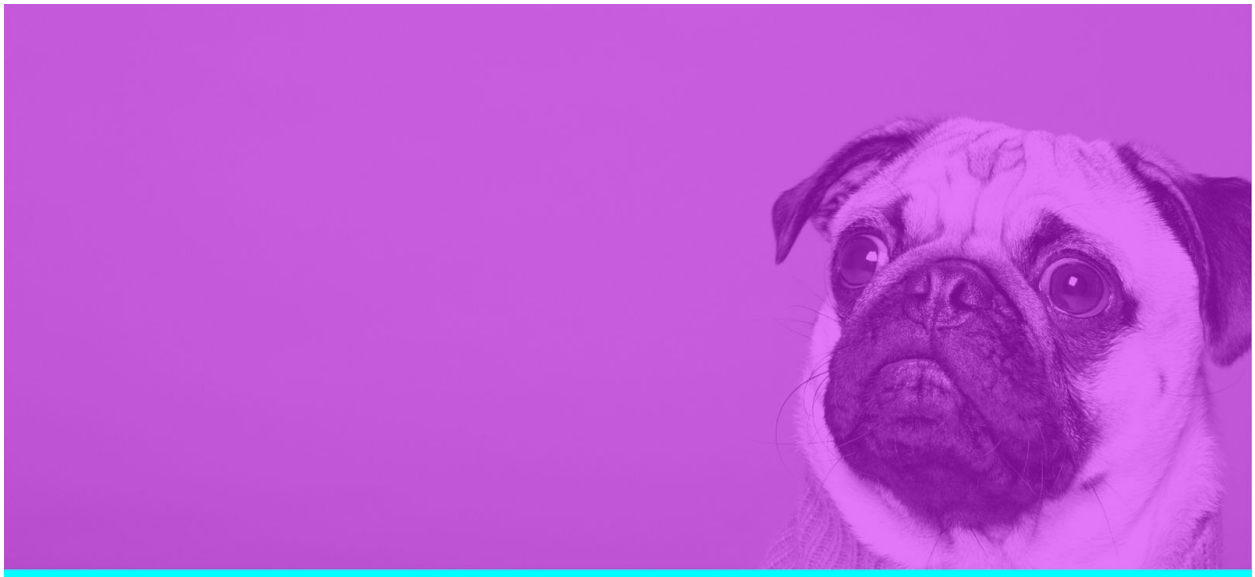
O czym jest ta książka?

Moim celem nie jest uczyć Cię dobrych praktyk pisania tekstów technicznych, mierzyć się z rozterkami językowymi i dyskutować o tym czy stawianie przecinka w danym miejscu ma sens. Dlatego ta książka nie wchodzi zbytnio w szczegóły, tylko w zwięzły sposób przedstawia aspekty dokumentacji, które wydają się istotne z punktu widzenia osoby zarządzającej nią w organizacji. Czytając ją, dowiesz się:

- Po co i komu jest potrzebna dokumentacja
- Jak kalkulować koszty i zwroty z inwestycji
- Jak przeanalizować potrzeby dokumentacyjne w produkcie (lub firmie)
- Kto powinien się zająć dokumentacją (umiejętności) i jak podzielić pracę (zespół)
- Jak zorganizować pracę zespołu dokumentacyjnego
- Jak mierzyć jakość i sukces dokumentacji

- Jak poprawiać istniejącą dokumentację
- Jaki jest cykl życia dokumentacji
- Jakich narzędzi używać

Po przeczytaniu tej książki możesz pogłębić wiedzę na poszczególne tematy lub znaleźć ludzi, którzy zajmą się szczegółami.



Po co nam dokumentacja?

Dokumentacja jest dla użytkownika.

Jeśli to wiesz i nie potrzebujesz, żebym Cię przekonywał, przejdź do następnego rozdziału.

Jeżeli uważasz, że dokumentacja jest niepotrzebna, to być może masz rację. Nie zawsze jest potrzebna. Jednak zachęcam Cię do czytania dalej.

Jeżeli chcesz przekonać swoje szefostwo, ale nie wiesz jak, to zdecydowanie czytaj dalej.

Tak naprawdę, jest wiele rodzajów dokumentacji i trochę tu upraszczam. Dokumentacja projektowa (wymagania, sprawozdania), biznesowa (umowy, aneksy do umów, aneksy do aneksów), czy firmowa (“minutki” ze spotkań, maile) to jedna sprawa. Ale tutaj piszemy o **dokumentacji użytkownika**, czyli takiej, która **pozwala użytkownikowi używać Twojego produktu**.

Jeżeli Twoja aplikacja, jest wystarczająco prosta i ma dobrze zaprojektowany interfejs, to prawdopodobnie nie potrzebuje dokumentacji. Ale jeżeli:

- Tworzysz coś bardziej skomplikowanego, gdzie jest wiele możliwych scenariuszy użycia i odkrycie ich wszystkich wymaga znajomości aplikacji
- Tworzysz coś, czego używa się pisząc kod
- Tworzysz intuicyjną aplikację dla konsumentów, ale zarabiasz na firmach, które jej

używają

...to potrzebujesz dokumentacji dla użytkowników.

Ryc. 1: Prosta aplikacja

Skomplikowana aplikacja

Większość aplikacji dla przedsiębiorstw to "skomplikowane aplikacje". Nawet jeśli interfejs jest zupełnie intuicyjny, użytkownik powinien robić więcej niż to, co przypadkiem odkryje. Poza tym oszczędzi czas, jeśli napiszemy mu jak osiągnąć cel.

Przykładem takiej aplikacji może być Microsoft Word. Kiedy zaczynam pisać, widzę kontrolki, które pozwalają mi zmienić wielkość czcionki i pogrubienie. Więc zaczynam: "Rozdział 1". Zwiększam czcionkę do 24, dodaję pogrubienie. Potem parę razy enter, żeby był odstęp. Zmniejszam czcionkę do 10, usuwam pogrubienie. "Pewnego razu, Marcin poszedł na ryby..."

Fajnie mi się pisze. Marcin ma różne przygody na rybach, ale teraz pora wprowadzić drugiego bohatera, Kamila. Najlepiej w nowym rozdziale.

Parę razy naciskam enter, żeby tekst zaczynał się na nowej stronie. Czcionka na 24,

pogrubienie. “Rozdział 2”.

Parę razy enter. Czcionka na 9. Nie, chyba było 10? Tak, 10! Piszę dalej.

Sporo tego mam, przydałby się spis treści. Ale spoko, Word jest intuicyjny, więc znajduję opcję Wstaw, potem Spis treści... Wstawił się, ale jest pusty.

Jaki popełniłem błąd? Trzeba było użyć stylów. Tytuł rozdziału oznaczyć jako Nagłówek 1, a tekst rozdziału jako Normalny. Nie muszę pamiętać jakich wielkości czcionek używam. I jak dopiszę jakieś zdanie, to nie będę musiał poprawiać gdzie się ma zaczynać następna strona. A jak wstawię spis treści, to nagłówki i numery stron się same wygenerują.

Taka jest dobra praktyka pracy w Wordzie. Ale skąd miałbym to wiedzieć?

Mógłbym się dowiedzieć z dokumentacji -e strony “Jak zacząć pisać w Wordzie”, albo “Jak napisać powieść w Wordzie”. (O nagłówkach powiemy sobie później.)

Aplikacja dla programistów

Przykładem tutaj może być dowolna implementacja Domain Object Model.

Jeżeli to zbyt techniczne dla Ciebie, pomiń dwa następne akapity i czytaj dalej od strzałki.

Wyobraź sobie, że nic nie wiem o tym jak działa DOM i muszę dojść do tego z opisów klas i metod. Chcę przejść przez dokument XML i zamienić taga “data wydania” na atrybut. Zaczynam od wczytania dokumentu, potem próbuję znaleźć wszystkie tagi “data wydania” i nadpisać ich wartość. Nie wiem jeszcze jak stworzyć atrybut, czy mogę nadpisać w miejscu czy muszę zrobić kopię i zapisać moje zmiany. Nie mówiąc już o dostępnych narzędziach i pomocnych metodach, wydajności różnych readerów, używaniu XPath albo XSL...

Wskakuję na StackOverflow i kopiuję stamtąd bloki kodu. Uczę się przy okazji metod, które pojawiają się w wynikach Google jako **pierwsze**, co może znaczyć, że są **najstarsze**.

Jako twórca tej implementacji DOM pewnie chciałbyś, żebym skorzystał z najnowszych funkcjonalności, ale skąd ja mogę o nich wiedzieć?

Jeżeli tworzysz bibliotekę, framework, lub API, to na pewno rozumiesz dlaczego dokumentacja jest NIEODZOWNA. Nikt nie zgadnie jaki kod pisać, żeby działało. Możesz im udostępnić swoje źródła i liczyć na to, że z nich się nauczą, ale w większości przypadków to nie jest praktyczne.

Dokumentacja z komentarzy w kodzie jest ok. Zwłaszcza, jeżeli pokazuje się w

zintegrowanym środowisku programistycznym (IDE) Twoich użytkowników. Ale nie zawsze wystarczy, żeby wytłumaczyć jak coś ma działać. A zwłaszcza jak zacząć.

Napisz kilka stron o tym jak zacząć i jak wdrożyć typowe funkcjonalności. Nawet jeśli większość użytkowników przeczyta tylko próbki kodu, to i tak im pomogłeś. Poza tym słowa wokół próbek też się komuś przydadzą.

Zaplecze dla użytkowników komercyjnych

Niektóre aplikacje nie wymagają instrukcji. Użytkownicy po prostu zaczynają ich używać i powoli stają się w tym dobrzy. A ty zarabiasz na tym, że w Twojej aplikacji biorą też udział firmy, które na tych użytkowników polują.

Przykładem takiej aplikacji w moich czasach (nie wiem, kiedy to czytasz) jest Snapchat.

Ludzie na Snapchacie tworzą swoje "historyjki", czyli posty które składają się ze zdjęć lub filmików z opisami. Każda historyjka jest dostępna przez kilka godzin, a potem znika na zawsze. Dlaczego ktoś tego używa? Bo to fajne.

Na Snapchacie jest wielu użytkowników (w moich czasach!), więc pojawiają się firmy, które chcą do tych użytkowników trafić ze swoimi produktami. I na tych firmach Snapchat zarabia.

Pracownik firmy, która reklamuje się na Snapchacie chce używać Snapchata prawidłowo, zgodnie z zamiarem autorów aplikacji. Do tego chce go używać tak, jak użytkownicy, którzy sami odkryli sekrety Snapchata. Dlatego właśnie ta aplikacja ma bardzo rozległą i szczegółową dokumentację.

99% użytkowników nawet nie będzie wiedziało, że ta dokumentacja istnieje. Ale ten 1% który płaci, na pewno z niej skorzysta.

Opłacalność dokumentacji

Skoro robimy coś w pracy, robimy to za pieniądze. Dlatego musimy sobie obliczyć Return on Investment (ROI). Innymi słowy, musimy ustalić jakim nakładem pracy opłaca się tworzyć dokumentację. Może się okazać, że jeszcze nie mamy pieniędzy, żeby się nią zająć, że opłaci się ona na innym etapie, albo, że nie opłaci się nigdy.

Zadanie jest z pozoru proste: sprawdź czy koszt przewyższa zyski. O ile koszty można obliczyć łatwo, to z zyskami jest dużo trudniej. Dokumentacji jako takiej nie sprzedajemy, więc nie ma jednej kwoty, którą możemy wykorzystać w obliczeniach. Ale po kolei.

Koszty

Jeżeli mamy jakąś osobę do pisania dokumentacji i pisze ona całą dokumentację, to rachunek jest prosty: koszty pracownicze tej osoby.

Do tego doliczmy koszty licencji oprogramowania, szkoleń i wyjazdów. Prosta sprawa, wszystko jest w rachunkowości.

Jeżeli ta osoba pisze kilka dokumentacji do kilku produktów, to dalej jest w miarę prosto podzielić jej czas procentowo. Podobnie jeśli jest więcej niż jedna osoba. Zawsze jest to jakaś zgrubna kwota, która daje punkt odniesienia.

Czasem potrzebujemy dokumentacji w kilku językach. Koszt tłumaczenia łatwo określić, nawet jeśli jest różny w zależności od rodzaju dokumentacji. Rzeczy, które trudniej

tłumaczyć to:

- Obrazki z zapisanym w nich tekstem
- Filmy i animacje
- Źle napisany tekst - co to znaczy “źle”? Spytaj technical writera lub tłumacza. Powiedzą Ci jak pisać, żeby było łatwiej tłumaczyć.

Koszty dokumentacji to raczej nie jest problem. Trudniej obliczyć zwrot z inwestycji. Dlaczego?

Kalkulowanie zwrotu z inwestycji

Zwykle dokumentacji nie sprzedajemy osobno, tylko razem z oprogramowaniem, albo wręcz udostępniamy za darmo. Dla wszystkich, nawet tych, którzy nie zapłacili. Przez to kalkulacja ROI staje się trudna.

Koszt pisania dokumentacji zwraca się w następujące sposoby:

- Przyciąga klientów - zanim ktoś kupi skomplikowany software, sprawdzi w dokumentacji jakie ma możliwości, jak go używać, z czym się integruje, i tym podobne. Sprawdź w analityce ilu użytkowników przechodzi z dokumentacji do zakupu (albo do cennika, albo do “skontaktuj się z nami”...)
- Zmniejsza koszty wsparcia technicznego - czy pracownicy helpdesku mogą odesłać użytkowników do dokumentacji? Jak często to robią? Czy widzisz zmianę w liczbie zapytań do supportu w miarę tego, jak rozwijasz dokumentację?
- Sprawia, że użytkownicy lepiej używają produktu - a przez to są z niego bardziej zadowoleni. Trudno to stwierdzić, ale zadowolony klient zostanie z Tobą dłużej.

Czy to znaczy, że możemy na tej podstawie obliczyć zwrot z inwestycji? Tak, ale każdy obszar niesie ze sobą problemy.

- Przyciąga klientów - ale nie każdy klient idzie bezpośrednio z dokumentacji do zakupu. Często między analizą a zakupem minie wiele miesięcy i dokona go inna osoba z firmy. Inżynier sprawdza funkcjonalność, a dział zakupowy kupuje.
- Zmniejsza koszty wsparcia technicznego - ale nie zawsze wiesz ilu ludzi się z Tobą nie skontaktowało, bo nie mieli potrzeby. Łatwiej wykazać, że sytuacja się zmienia, ale czy to znaczy, że masz zacząć od kiepskiej dokumentacji i powoli ją poprawiać? Być może.
- Sprawia, że użytkownicy lepiej używają produktu - ale jeśli są zadowoleni i wszystko idzie dobrze, to pewnie o tym nie wiesz. Wiesz, jeżeli narzekają. A jeśli nie narzekają, to może jest super, a może Twoje oprogramowanie leży w kącie i zbiera kurz.
- Ułatwia utrzymanie wiedzy w organizacji. Kiedy przychodzi ktoś nowy, może się wdrożyć z pomocą dokumentacji. Kiedy ktoś odchodzi, nie tracimy całej jego

wiedzy.

Właśnie z powodu tych trudności wiele firm nie ma możliwości obliczyć ROI z dokumentacji.

Być może właśnie Ty jesteś w sytuacji, w której jesteś w stanie to obliczyć. Jeżeli tak, to masz szczęście. Korzystaj z tego!

Zrozumiały koszt

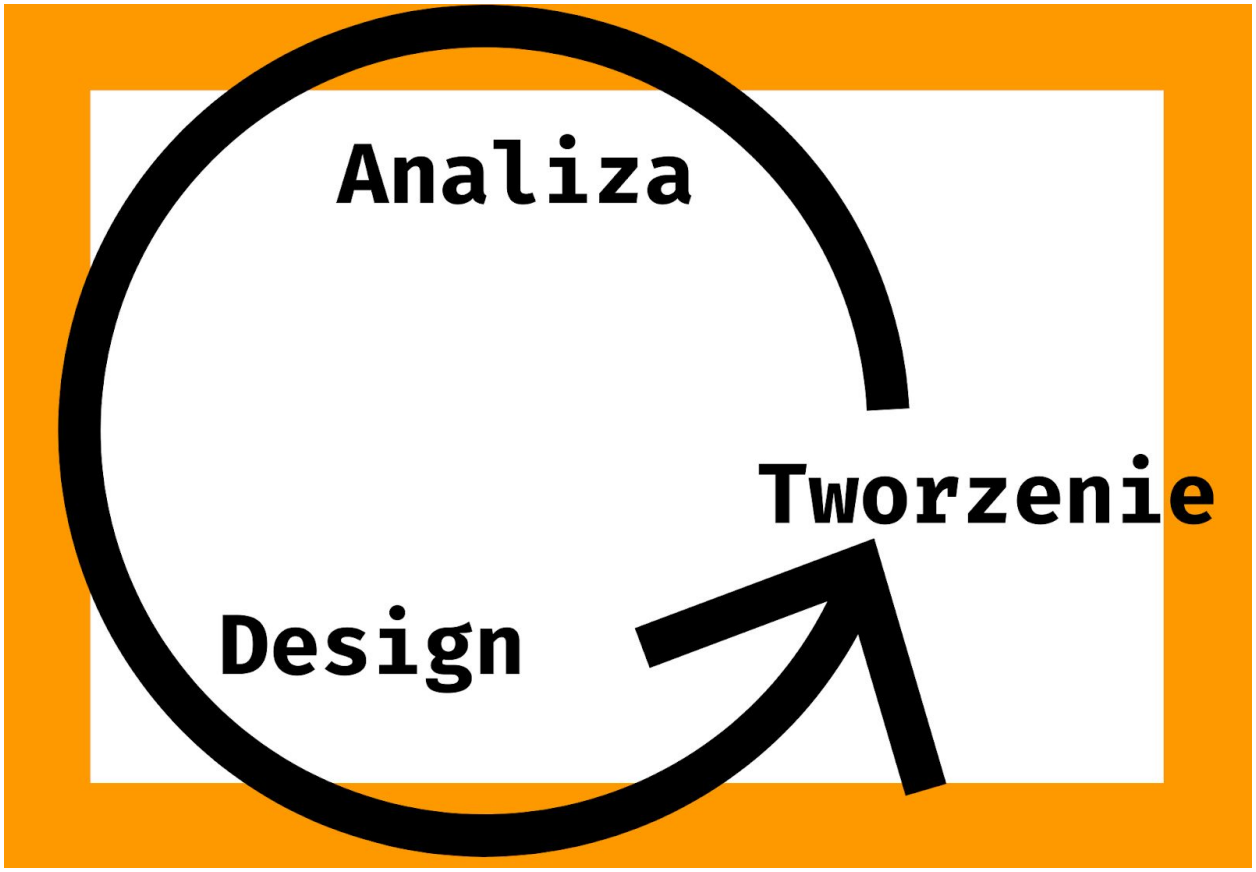
Ponieważ tak trudno jest obliczyć zwrot z inwestycji w dokumentację, wiele firm traktuje ją jako stały koszt wytwarzania oprogramowania. Musisz napisać kod, musisz przetestować aplikację i musisz napisać dokumentację.

W takich sytuacjach gra toczy się o utrzymanie rozsądnej równowagi zysków i strat. Ważne staje się wtedy kto pisze dokumentację (czy robi to programista?), oraz ile jej jest. Ale tym zajmiemy się później. Najpierw musimy wiedzieć co trzeba pisać.

Analiza, design, tworzenie

Skup się na odbiorcy.

Wyobraź sobie, że kroki przed nami, to:



Zatrzymajmy się na tym. Chodzi na razie o to, żeby przejść cykle analizy i designu[1] aż będziemy gotowi zacząć tworzyć. Czyli, między innymi, pisać. Te trzy kroki wystarczą na początek.

Nie skupimy się na tym co potem: utrzymujemy treść, dostajemy feedback, analizujemy feedback, designujemy rozwiązanie, wdrażamy je, i tak dalej. Ten cykl się nigdy nie kończy.

Analiza to nie design

Analiza jest obiektywna. Analizujesz co chcesz osiągnąć. Innymi słowy, kładziesz na stół wszystkie cele. Jakie są Twoje cele? Spełnić **potrzeby** użytkownika.

Na etapie analizy nie proponuj rozwiązań, od tego jest design.

Żeby przeprowadzić analizę, skup się na użytkowniku, nie na produkcie. Nie opisuj funkcjonalności produktu, bo tam są. Opisuj to, co użytkownik potrzebuje osiągnąć.

[1] Dlaczego nie "projektowanie" lub "projekt"? Oba te słowa są bardzo ogólne. W tej książce nie staram się sztucznie unikać anglicyzmów, których używamy w branży i design jest tego bardzo dobrym przykładem: to konkretne słowo, którego akurat potrzebujemy.

Żeby zebrać potrzeby użytkownika, przeprowadź wywiady i skonstruuj persony. Możesz o tym przeczytać wszędzie, albo poradzić się UX designera, dlatego nie będziemy się nad tym rozwodzić. Najważniejsze co musisz pamiętać, to rozwinięcie skrótu UX. Bo analizując potrzeby dokumentacyjne, analizujesz całe user experience - doświadczenie użytkownika podczas korzystania z produktu.

W wyniku przeprowadzenia wywiadów i stworzenia person jesteśmy w stanie zidentyfikować potrzeby użytkownika, które mogą mieć następującą formę:

- Użytkownik musi zainstalować wtyczki w gniazdkach i podłączyć je do wifi
- Użytkownik chce ustawić termostat tak, żeby podniósł temperaturę zanim wstanie z pracy i ekspres tak, żeby zrobił kawę do śniadania
- Użytkownik chce, żeby alarm ignorował kota, który szwęda się po domu

Design pod konkretne potrzeby

Teraz, kiedy na stole leżą potrzeby, spełnijmy je za pomocą dokumentacji.

Najbardziej typowa dokumentacja to “strona z pomocą”, a więc na etapie designu tworzymy jej mapę (sitemap). Czyli listę podstron.

Na podstawie potrzeb, które wynikają z analizy, zrób listę nagłówków. Zamknij się w pokoju z potencjalnymi użytkownikami i poukładajcie razem te nagłówki w sensowną kolejność. Może pogrupujcie je w większe strony, wtedy będzie łatwiej znaleźć je w nawigacji. Na przykład:

Strona	**Szczegółowe tytuły**
Zainstaluj	Zainstaluj sensory Zainstaluj kontrolę napięcia Zainstaluj oprogramowanie ...
Skonfiguruj	Dodaj użytkownika Usuń użytkownika Skonfiguruj tryb nocny ...
Używaj	Ustaw timer na ekspresie do kawy Ustaw budzenie muzyką Dostosuj termostat do trybu dobowego ...
Dodatkowe funkcjonalności	Głośniki w trybie “follow”

	Tymczasowy dostęp do zamka w drzwiach Pomijanie kota ...
Parametry i komendy	Parametry techniczne Wymagania sprzętowe Komendy głosowe Kody błędów ...

Zauważ, że nagłówki napisane są w formie "zadań" do wykonania. Piszemy z punktu widzenia użytkownika i to jego potrzeby są najważniejsze. W dokumentacji technicznej często stosuje się podejście "zadaniowe", bo użytkownik nie czyta dokumentacji tak jak czyta się powieść. Użytkownik próbuje **coś zrobić**. (Najczęściej rozwiązać jakiś problem.)

Ostatnia grupa to informacje podręczne, czyli takie, do których użytkownik wraca, żeby coś sprawdzić. Na przykład, znajdzie tu listę komend lub kodów błędów. Do tej strony użytkownik zapewne będzie wracał częściej niż do innych.

Wracamy do analizy

Jak już masz design, wróć do kroku analizy i przetestuj go. Posadź użytkownika przez zadaniem i daj mu listę nagłówków. Zapytaj go: "gdzie poszukasz rozwiązania?" Użytkownik nie będzie mógł znaleźć odpowiedniej sekcji - to dobrze. Zapytaj gdzie by jej szukał i tam ją umieść.

Musisz skonfrontować swój design z grupą trzech do pięciu użytkowników. Nie daje to wyników statystycznych, ale wskazuje trendy.

Uważaj jednak, żeby nie przedobrzyć. Nie spędzaj za dużo czasu na cyklu analiza-design, bo zamkniesz się w martwym kręgu. Pamiętaj, że projektujemy dokumentację a nie zastawkę serca. Jak mówią za oceanem: *good enough is good enough*.

Tworzenie, czyli nie tylko pisanie

Mówi się, że dokumentację się pisze. Jednak rozwój technologiczny sprawił, że tego stwierdzenia nie można traktować dosłownie. Istnieją inne formy przekazu, które mogą sprawdzić się lepiej niż słowo pisane. Twoja analiza i design pokaże co tak naprawdę trzeba zrobić. Może zamiast pisać lepiej będzie stworzyć rysunki, animacje, lub filmiki z tutorialami. Pisanie jest tak naprawdę najtańszym rozwiązaniem, szczególnie jeśli tłumaczysz swoją dokumentację na inne języki. Jeśli jednak zdecydujesz się dodać jakieś multimedia, rozważ wcześniej koszty tłumaczenia.

Kto tworzy dokumentację?

W fazie wdrożenia, o której była mowa w rodziale Planowanie, analiza, design, zabieramy się za pisanie (czy też tworzenie treści w innej formie, w zależności od tego co ustaliliśmy podczas analizy i designu). Czujemy trochę, że pisanie to sporo pracy, dlatego zanim na dobre się tym zajmiemy, zastanówmy się kto powinien zajmować się tworzeniem dokumentacji. W zależności od skali przedsięwzięcia, naszą dokumentacją może zajmować się jeden człowiek albo cały zespół.

Role związane z dokumentacją

W procesie tworzenia dokumentacji do obsadzenia są następujące role:

1. Ktoś ustala wymagania dotyczące dokumentacji.
2. Ktoś pisze dokumentację.
3. Ktoś ją opiniuje.
4. Ktoś ją publikuje (potrzebna jest strona z dokumentacją lub coś podobnego).

Wymagane umiejętności i zasoby

Każda rola wymaga odpowiedniego zestawu umiejętności i zasobów.

Dla kogoś, kto ustala wymagania dotyczące dokumentacji

Taka osoba musi znać potrzeby użytkowników. Najlepiej, żeby ich nie wymyślała, ani nie zgadywała, tylko je zbadała.

Wymagane umiejętności:

- Przeprowadzanie badań i wywiadów (często stosowane w UX designie)
- Analiza wyników badań
- Priorytetyzacja wymagań

Wymagane zasoby:

- Dostęp do użytkowników

Dla kogoś, kto pisze dokumentację

Wymagane umiejętności:

- Jasne i zwięzłe pisanie
- Pisanie dla konkretnej publiki
- Znajomość produktu

Wymagane zasoby:

- Dostęp do wyników badań potrzeb użytkownika
- Dostęp do produktu, żeby testować dokumentację
- Dostęp do ekspertów, żeby wyjaśniać wątpliwości i zawiłości techniczne
- Dostęp do edytora - kogoś kto zrobi korektę, poprawi styl, i zadba o spójność przekazu pochodzącego od różnych autorów.

Dla kogoś, kto opiniuje dokumentację

Wymagane umiejętności:

- Znajomość produktu, żeby potwierdzić, czy informacje w dokumentacji są prawdziwe

Wymagane zasoby:

- Dostęp do produktu, żeby testować dokumentację
- Dostęp do wyników badań potrzeb użytkownika, żeby zweryfikować czy dokumentacja spełnia te potrzeby
- Dostęp do ekspertów, żeby wyjaśniać wątpliwości i zawiłości techniczne, które mogą wyniknąć podczas testowania.

Dla kogoś, kto publikuje dokumentację

Wymagane umiejętności:

- Tworzenie stron/portali/książek, czy innych form przekazu dokumentacji
- Tworzenie zautomatyzowanych buildów i deploymentów, CI/CD, automatyczne testy weryfikacyjne, itp.

Wymagane zasoby:

- Dostęp do odpowiednich narzędzi i technologii (na przykład CCMSa z możliwościami publikacji)
- Dostęp do zespołów wspierających infrastrukturę i procesy dostarczania oprogramowania w celu uzyskania wsparcia i stosowania dobrych praktyk

Potencjalni kandydaci

Poszczególne role mogą być obsadzone przez następujące osoby.

Programista

Tworzy aplikację i przez to zna ją najlepiej. Ale czy pisanie dokumentacji to jest najlepszy użytek jego czasu? Programistów zatrudnia się po to, żeby pisali kod, bo na tym najlepiej się znają. W tym obszarze programiści pracują najbardziej wydajnie, więc czas ich pracy jest dobrze spożytkowany. Biorąc pod uwagę wysokość ich zarobków, warto się zastanowić czy nie lepiej oddać tworzenie dokumentacji osobie wyspecjalizowanej właśnie w tej dziedzinie.

Poza tym, ktoś kto tworzy aplikację może mieć inną perspektywę niż użytkownik. Programiście bardzo dużo czasu zajęło wdrożenie odpowiedniej wersji SNMP, albo zgodności ze standardem X, Y, lub Z. Naturalnie w jego głowie ta sprawa wydaje się bardzo ważna, podczas gdy może nie ma w ogóle znaczenia dla użytkownika.

Wyjątkiem jest oprogramowanie dla programistów - wtedy programista naturalnie ma odpowiednią perspektywę.

Analityk biznesowy

Ma kontakt z klientem i zna jego potrzeby. Na ich podstawie stworzył plan wdrożenia oprogramowania. Tylko jego czas zwykle dzieli się pomiędzy bardzo dużą liczbę spotkań. Tam leżą talenty analityka biznesowego, więc tam je wykorzystujemy.

W związku z tym, czasem pojawia się następujący model: analityk biznesowy tworzy wygania dokumentacji i organizuje programistów, którzy ją napiszą.

UX designer i UX writer

Pracuje nad optymalnymi interakcjami użytkownika z oprogramowaniem. Zwykle projektuje interfejs, czasem pisze teksty, które pokażą się w tym interfejsie. Może wydawać się naturalne, że ta osoba opracuje przynajmniej część dokumentacji użytkownika. A mimo to, nigdy się z tym nie spotkałem. Być może dlatego, że projektowanie interfejsu to zupełnie inny zestaw umiejętności niż projektowanie treści technicznych.

Technical writer

Specjalizuje się w projektowaniu i pisaniu dokumentacji użytkownika. Dużo rozmawia z programistami i z innymi osobami w firmie, testuje, używa oprogramowania, wciela się w różne role. Czasem pisze[2].

Dlaczego potrzebna jest ta dodatkowa osoba?

- Skupia się tylko na dokumentacji, ma więc siłę specjalizacji. Dzięki temu zna praktyki, narzędzia i procesy związane z dokumentacją
- Zna szerszy kontekst niż osoby, które bezpośrednio tworzą produkt; zna całą funkcjonalność i cele biznesowe użytkowników

Jeżeli masz okazję pracować z technical writerami, to większość problemów spocznie na ich barkach. Musisz tylko zadbać o odpowiednie środowisko pracy dla nich.

Niestety, duże zespoły technical writerskie mają tendencję do rozrastania się w nieskończoność i dokumentacja zaczyna żyć własnym życiem, które niestety z biegiem czasu ma coraz mniej wspólnego z życiem produktu.

[2] Wbrew powszechnemu przekonaniu, technical writer spędza relatywnie mało czasu na właściwym pisaniu. Większość dnia, szacunkowo ok. 80%, wypełniają mu czynności takie jak szukanie informacji i wyciąganie wiedzy od ekspertów.

Zespół dokumentacyjny

Zespół zajmujący się dokumentacją, to grupa ludzi, którzy znają się na pisaniu i publikowaniu. Potrafią zrekrutować nowe osoby, bo wiedzą gdzie szukać podobnych sobie. Mogą też utrzymywać swoje wyspecjalizowane narzędzia.

Taki zespół może istnieć w firmie na dwa sposoby: w modelu zintegrowanym lub agencyjnym.

Model zintegrowany

W tym modelu, zespół dokumentacyjny dostarcza specjalistów, którzy dołączają do zespołów projektowych. Wtedy technical writerzy dołączają do innych działów i w nich pracują, a zespół pełni tylko role administracyjne i szkoleniowe.

Dlaczego nie zatrudnić writerów bezpośrednio w zespołach?

- Bo może nie są potrzebni na stałe - wtedy można ich rotować między zespołami
- Bo zespół wytwarzający oprogramowanie zna się na oprogramowaniu i może mu być trudno znaleźć, zatrudnić i wyszkolić technical writera

Wiem, że fajnie jest przeczytać trzy podpunkty, ale mogę znaleźć tylko dwa.

Zaletą tego modelu jest to, że technical writer obcuje z produktem na co dzień, więc gromadzi wiedzę, która inaczej byłaby nieosiągalna. Tego typu model sprawdza się w

wypadku skomplikowanych produktów.

Model agencyjny

W tym modelu, zespół dokumentacyjny przyjmuje “zlecenia” na wykonanie dokumentów. Zlecenia przychodzą od managerów produktów i zespół oddelegowuje jedną lub więcej osób, żeby wypełnić zlecenie. Agencja może być częścią firmy, lub zewnętrznym usługodawcą.

Często zlecenie rusza pod koniec cyklu produkcyjnego, to znaczy na przykład miesiąc przed końcem projektu trwającego rok. Technical writerzy przydzieleni do zadania przeprowadzają wtedy wywiady z kim trzeba, projektują dokument, opiniują, piszą, znowu opiniują, publikują i zostawiają sprawę do czasu, kiedy zostaną wezwani ponownie.

W tym samym modelu technical writer może też być przypisany do kilku projektów na stałe. Wtedy ma większą wiedzę o każdym z produktów i ich odbiorcach, ale nie ma na tyle pracy, żeby poświęcić cały czas jednemu z nich. Technical writer nie jest uważany za część zespołu a odpowiedzialność i przydział czasu zależy od jego managera.

Który model wybrać?

Bardzo dużo zależy od cyklu projektu.

Jeżeli projekt trwa długo, na przykład 6 miesięcy lub dłużej, to model agencyjny może sprawdzić się dobrze. Zwłaszcza, jeżeli przez większość czasu nie ma zmian, które powinny się znaleźć w dokumentacji.

Na przykład, zespół programistów pracuje nad modelem danych, migruje do nowej bazy i poprawia infrastrukturę, ale nie ma to wpływu na interfejs użytkownika. Dopiero pod koniec cyklu zespół może dodawać nowe funkcjonalności dla użytkownika i nie doda ich bardzo dużo. Dokumentacja dla użytkownika nie potrzebuje uwagi przez ten cały czas, a technical writer w zespole nie miałby nic do roboty.

Jeżeli zespół pracuje w krótkich sprintach, to technical writer musi się stać członkiem zespołu, żeby móc szybko reagować na zmiany.

Na przykład, sprint trwa dwa tygodnie i na koniec zespół zmienił działanie czterech różnych komponentów produktu. Technical writer z zewnątrz może przyjść na demo i zobaczyć zmiany, a potem nanieść je w dokumentacji. Tylko niestety demo jest tuż przed publikacją, więc musimy się liczyć z tym, że dokumentacja będzie przez jakiś czas nieaktualna.

Oczywiście zespół mógłby w trakcie pracy dawać znać zewnętrznemu writerowi jakie zmiany nadchodzą, ale to będzie kosztowało ich czas. A poza tym do samego końca nie będzie wiadomo jaki jest końcowy efekt ich pracy.

Jest wiele innych czynników decydujących o wyborze. Żeby nie marnować zbyt wielu akapitów na wijące się dyskusje, wypiszę kilka z nich w podpunktach:

- Kultura wymiany wiedzy w firmie
- Budżet
- Poziom skomplikowania produktu
- Dostępność zewnętrznych agencji
- Dostępność technical writerów do rekrutacji
- Wartość biznesowa dokumentacji technicznej
- Rozmieszczenie członków zespołu - agencja może się lepiej sprawdzić w koordynacji zespołów rozproszonych po świecie

Metryki jakości

Jakość powinno się dać zmierzyć po to, żeby móc ją utrzymać lub podnieść. Ale jak mierzyć jakość dzieła sztuki (bo takim właśnie dziełem jest dokumentacja). No dobra, to był żart, ale jednak jest w tym stwierdzeniu ziarno prawdy.

Kiedy myślimy o jakości dokumentacji, do głowy przychodzą następujące metryki:

- Liczba defektów w dokumentacji. Czyli ktoś zgłasza, że informacja w dokumentacji jest nieprawdziwa, lub brakuje jakiejś informacji.
- Liczba zapytań do supportu, które powinny być rozwiązane w dokumentacji, ale nie są. Nie należy mylić tego z liczbą zapytań do supportu, które są rozwiązane w dokumentacji, ale użytkownicy o tym nie wiedzą. To także jest problem z dokumentacją, ale jest on bardziej związany z jej dostępnością, wyszukiwarką, nawigacją, itp.
- Liczba zapytań do działu sprzedaży, które pochodzą z dokumentacji.
- Indeks czytelności, czyli jak łatwo jest czytać dokumentację. Na przykład: indeks Gunninga, Flesch-Kincaid, SMOG, Coleman-Liau. Im niższy, tym lepszy.

Wadliwa dokumentacja

Gdy czegoś brakuje, lub informacje są nieprawdziwe, dowiemy się tego od użytkowników. Popularną metodą zbierania tego typu informacji jest możliwość komentowania dokumentów. Komentarze nie muszą być publicznie widoczne, najważniejsze, żeby zespół

dokumentacji na nie reagował.

Komentarze od użytkowników są bardzo cenne, bo tylko jeden na miliard użytkowników wystawi komentarz (statystyki niepotwierdzone). Z tego część będzie dotyczyć dokumentacji, a część samego produktu. Na obie informacje warto zareagować - odpisać i poprawić to, co jest nie tak. Użytkownik poczuje się bardzo dobrze, jeśli dostanie od nas prostą i przyjazną wiadomość - nie z wymówkami, ale ze szczerym podziękowaniem za poświęcony czas i z opisem kroków jakie podjęliśmy w sprawie zgłoszonego przez niego problemu. Chodzi o coś takiego.

> *Cześć Marek,*
>
> *Dziękujemy za komentarz. Napisałeś, że w tabeli brakuje wartości dla parametru premFIK. Już dodaliśmy brakujące informacje. Daj nam znać, jeśli zauważysz coś jeszcze.*
>
> *Pozdrawiam,*
>
> *Piotrek Jakiśtam*
>
> *Technical writer, SomeSoftwareSystems.io*

Dzięki takiej wiadomości zyskujemy dwie rzeczy: zaufanie i nowego kontrolera jakości.

Żeby mierzyć ten wymiar jakości, zbierajmy komentarze i dzielmy je na kategorie. Na przykład, zgłoszenia dotyczące dokumentacji, zgłoszenia dotyczące produktu, wyrazy frustracji, chęć udowodnienia swojej wyższości intelektualnej, itp.

Raz na rok (kwartał, miesiąc, zależy od ilości) możemy porównać wyniki z poprzednim okresem i przyjrzeć się komentarzom w każdej kategorii. Czy poprawiliśmy dokumentację lub produkt? Czy klient został utrzymany? Czy liczba komentarzy się zmienia?

Rezultaty takiego procesu mogą być ogromną kopalnią wiedzy.

Dokumentacja jako support

W idealnym świecie, nasi użytkownicy znaleźliby rozwiązania swoich problemów sami. Dzięki temu szybciej osiągneliby swój cel i byliby bardziej zadowoleni, a my oszczędzilibyśmy na kosztach utrzymania helpdesku.

Jako metrykę możemy przyjąć liczbę zapytań do supportu. Jeżeli liczba spada kiedy poprawiamy dokumentację, to dobrze. Ale trudno powiązać tę liczbę jednoznacznie z dokumentacją, bo być może poprawiliśmy produkt, albo mamy mniej klientów.

Moglibyśmy też dodać w dokumentacji przycisk “skontaktuj się z supportem” i liczyć ile osób go używa. Jeżeli przycisk jest w tym samym miejscu i tak samo dostępny, a liczba

maleje, to może być znak, że robimy coś dobrze. Naszym celem może być ciągłe zmniejszanie tej liczby. Żeby to zrobić, analizujemy jaką ścieżkę przeszedł klient i reagujemy na powtarzające się potrzeby. Na przykład:

Nasza aplikacja integruje się z komunikatorem Skype. Mamy dużo zapytań do supportu o to, czy integruje się ze Skype for Business i Microsoft Lync. Widzimy, że niektórzy użytkownicy szukali słów "Skype for Business" i "Microsoft Lync" na stronie z dokumentacją, lub przeglądali sekcję "Integracja z komunikatorami", a potem klikali "skontaktuj się z helpdeskiem".

Patrzymy do dokumentacji i widzimy, że w sekcji integracja z komunikatorami napisaliśmy o Skypie, ale nie wspomnieliśmy o pozostałych dwóch. Dodajemy sekcje o brakujących komunikatorach i rozjaśniamy sprawę

Jeżeli w dokumentacji mamy link typu "skontaktuj się z helpdeskiem", to możemy mierzyć jak wiele osób go używa.

Jeżeli podejdziemy do sprawy nie metrykowo, tylko jakościowo, to możemy analizować zapytania i patrzeć na następujące rzeczy:

- Czy zapytanie mogłaby rozwiązać dokumentacja - jeżeli tak, to dodajmy brakujące informacje.
- Czy zapytanie jest rozwiązane w dokumentacji - jeżeli tak, to znajdźmy odpowiedź na pytanie dlaczego klient z niej nie skorzystał.
- Czy zapytanie się powtarza - zastanówmy się czy warto poprawić produkt.

Na dłuższą metę takie rozwiązanie może być lepsze niż śledzenie metryk, ale trudno zweryfikować jego skuteczność. Jeżeli ciągle pojawiają się te same zapytania, to znaczy, że nasze działania nie są skuteczne.

Dokumentacja jako narzędzie sprzedaży

Nasz potencjalny klient szuka oprogramowania, które najlepiej spełni jego cele. Przegląda ofertę naszą i konkurencji pod każdym względem, z czego najważniejszym pytaniem dla niego jest, "czy to oprogramowanie spełnia moje potrzeby?". Żeby odpowiedzieć na to pytanie, może sięgnąć do dokumentacji.

Jeżeli nasza dokumentacja jasno da mu na to odpowiedź, to być może kliknie na naszej stronie przycisk "oferta", albo "cennik", albo "skontaktuj się z działem sprzedaży". Możemy liczyć ile takich sytuacji miało miejsce. Potem możemy porównać ile razy ktoś kliknął "kupuj" ze strony z dokumentacją, a ile razy np. ze strony marketingowej. Wtedy pojawia się nowa metryka - ilość potencjalnych sprzedaży ("leadów") wygenerowanych przez dokumentację.

Możemy postawić sobie za cel, żeby zwiększać liczbę takich "leadów". Ale nawet sam fakt,

że one występują bardzo dobrze świadczy o naszej dokumentacji.

Poprawianie czytelności tekstu

Indeks czytelności jest miarą tego jak łatwo jest czytać naszą dokumentację. Przykładem może indeks Gunninga, Flesch-Kincaid, SMOG czy Coleman-Liau. W przypadku większości indeksów, im niższy, tym lepszy.

Ta metryka ma sens jeżeli nie jesteśmy w stanie powiedzieć nic więcej o jakości. Na przykład liczba defektów i zapytań jest na tyle niska, że nic nie oznacza. Wtedy traktujemy czytelność tekstu jako nasz własny sposób na utrzymanie poziomu jakości. Możemy też stosować ten indeks obok innych metryk.

Jest wiele aplikacji, które określi indeks czytelności naszych tekstów. Oprogramowanie analizuje liczbę słów na zdanie, zdań na akapit, i tym podobne. Bardziej zaawansowane programy patrzą na liczbę przysłówków, rzadko spotykanych słów, i tym podobne. Większość z nich, niestety, działa tylko w języku angielskim, ale w świecie oprogramowania to nie jest duży problem, ponieważ większość tekstów jest i tak pisana po angielsku.

Cykl życia dokumentacji

Oprócz cyklu w jakim nasza dokumentacja jest wytwarzana, istnieje jeszcze większy cykl, który określa jej cały żywot, od poczęcia aż po grób i życie pozagrobowe.

Dzieciństwo i dojrzewanie

Niezależnie od tego czy rozwijamy dokumentację zwinnie czy wodospadowo, na początku musi być niedojrzała. Nasza dokumentacja nie została jeszcze przeczytana przez klientów, więc nie mamy od nich feedbacku[3]. Nie opisaliśmy jeszcze wszystkiego, bo zaczęliśmy od najwyższych priorytetów. Nasz produkt jeszcze się rozwija, a dokumentacja wraz z nim.

Ten okres jest ważny, bo pozyskujemy nowych klientów, dlatego powinniśmy szybko reagować na potrzeby, odpowiadać personalnie i dokładać starań. W zależności od skali, nasz technical writer musi poświęcić dokumentacji cały swój czas, lub nasz zespół musi oddelegować do projektu więcej osób, lub zaangażować najlepszych.

Dorosłość

Teraz nasza dokumentacja i produkt są w miarę stabilne. Skupiamy się na utrzymaniu naszych treści. Naprawiamy głównie defekty i raczej nie dokonujemy jakichś

[3] Słowo “feedback”, to kolejne, obok “designu”, konkretne słowo, którego potrzebujemy. “Opinia” czy “informacja zwrotna” nie oddają w pełni znaczenia, dlatego zdecydowałem się po raz kolejny nie unikać anglicyzmów.

rewolucyjnych zmian. Czasem usuwamy treści dotyczące funkcjonalności, których już nie wspieramy. W tym okresie możemy przekierowywać naszych writerów do innych projektów.

Schyłek

W końcu nasz produkt przechodzi na emeryturę, a wraz z nim, dokumentacja. Napiszmy o tym! Jeżeli nie będziemy już wspierać produktu, to dajmy o tym znać użytkownikom, napiszmy dlaczego i podziękujmy im za lata współpracy.

Jeżeli zastępujemy produkt innym, to napiszmy użytkownikom dlaczego warto się przenieść, jakie będą mieli z tego korzyści. Napiszmy też jak skutecznie się przemigrować, i zapewnijmy, że wszystko będzie w porządku.

Narzędzia w dokumentacji

Model “tradycyjny”

W tym modelu stosuje się narzędzia ściśle związane ze światem komunikacji technicznej. Często poza technical writerami nikt o nich nie słyszał, nie wspominając już o tym, że nie umie oraz nie chce z nich korzystać. Nie dlatego, że są to złe rozwiązania, ale dlatego, że spełniają potrzeby w świecie, w którym zostały wymyślone i osoby funkcjonujące w innym świecie nie czują potrzeby ich używania.

Największą zaletą narzędzi typowo writerskich jest spora liczba funkcji, które dostajemy od razu i to, że są stworzone z myślą o osobach tworzących dokumentację. Dzięki temu możemy mocno zredukować zaangażowanie osób technicznych, np. programistów, w konfigurację i dodanie brakujących funkcji, ponieważ wiele rzeczy możemy zrobić sami. Dodatkowo, w pakiecie dostajemy wsparcie techniczne producenta co na pewno ułatwia rozwiązywanie problemów z narzędziem.

Z drugiej strony, musimy się liczyć z dość wysokim kosztem zakupu oraz z tym, że nie będziemy mieli szczegółowej wiedzy na temat tego jak działa narzędzie, przez co nie będziemy w stanie samodzielnie radzić sobie z pewnymi problemami oraz dodawać nowych funkcji w łatwy sposób. To z kolei może spowodować wydłużony czas potrzebny na wdrożenie jakiegoś rozwiązania oraz wygenerować dodatkowy koszt - np. będziemy musieli zapłacić producentowi za rozszerzenie narzędzia o to czego nam brakuje.

Ale może właśnie taki model nam pasuje, bo w zespole dokumentacyjnym nie mamy osób mocno technicznych i samodzielne utrzymywanie narzędzi nie jest możliwe.

Help Authoring Tool (HAT)

HAT chyba najlepiej określić jako kombajn, który pozwala nam na tworzenie dokumentacji od A do Z, czyli od pisania treści po jego publikację do żądanego formatu.

Przykładem takiego narzędzia jest MadCap Flare i Adobe RoboHelp. HAT zapisuje pliki w prostym repozytorium, albo na dysku, ale pozwala na utrzymanie wiele helpów z jednego zbioru topików. W tych topikach możemy stosować zmienne typu “nazwa produktu” i zmieniać ich wartość w zależności od tego, który help generujemy. HAT oferuje też możliwość wielokrotnego wykorzystania tej samej treści (reuse).

Jeżeli nasz produkt ma kilka wariacji, które mają części wspólne, ale używają ich różni użytkownicy, to warto zastosować reuse. Klasycznym przykładem ze świata poza oprogramowaniem jest pralka. Model A jest podstawowy, model B ma trzy dodatkowe funkcje, a model C ma jedną z tych trzech i dwie inne. Przykładem ze świata oprogramowania może być Word, PowerPoint i Excel, gdzie podstawowe operacje (zapisywanie plików, integracja z OneDrive, czy pisanie makr) są takie same.

Component Content Management System (CCMS) i standardy XML

Jeżeli przechowujemy bardzo dużo treści, mamy w niej dużo zależności (reuse, single sourcing, content references, itp.), to krokiem ponad HATy są CCMSy.

Component Content Management System (CCMS) łączy w sobie:

- Repozytorium i narzędzia do zarządzania wersjami
- narzędzia do zarządzania zadaniami i workflowem
- narzędzia do review
- system do zarządzania tłumaczeniami, może pamięć tłumaczeniową
- narzędzia do publikacji treści
- narzędzia do zarządzania jakością
- edytory i inne cuda na kiju

Tego typu system działa w połączeniu ze standardem przechowywania treści. Najpopularniejsze standardy na rynku to DITA, Docbook i S1000D. S1000D jest używany głównie w wojsku i lotnictwie (chociaż spotyka się go w szerszym przemyśle maszynowym). Docbook powoli umiera. Oprócz nich, istnieje jeszcze wiele niszowych standardów XMLowych, często związanych ściśle z konkretnym CCMSem.

DITA święci triumfy na rynku oprogramowania enterprise i istnieje wiele CCMSów, które z nią współpracują, a także kilka edytorów XML, które ułatwiają życie technical writerom, na przykład Oxygen (Syncro Soft), czy Adobe FrameMaker.

Model docs like code

W momencie kiedy piszę te słowa, na popularności mocno zyskuje (szczególnie wśród tech writerów współpracujących blisko z programistami) model utrzymywania dokumentacji o nazwie *docs like code* (znany też jako *docs as code)*. Oznacza to, że używamy podobnych narzędzi i procesów jak do tworzenia oprogramowania. Czyli:

- Źródła dokumentacji przechowujemy w systemie kontroli wersji (VCS), takim jak git czy SVN.
- Treść utrzymywana jest w plikach tekstowych, a piszemy ją używając języka znaczników, na przykład Markdown.
- Do pisania używamy edytora tekstowego, najlepiej takiego, który wspiera używany przez nas język znaczników. Przykładem może być Visual Studio Code albo Atom.
- Do publikacji używamy narzędzi CI typu Jenkins, Travis czy Teamcity
- Budujemy statyczne strony HTML za pomocą generatora stron statycznych, takiego jak Sphinx czy Docusaurus. Rzadko publikujemy PDFy.

To dobre podejście do iteracyjnego tworzenia dokumentacji, jeżeli pamiętamy o tym, żeby nie powtarzać tej samej informacji na wielu stronach. Zapytajcie technical writera jak łatwo o pomyłkę, jeśli utrzymujemy tę samą informację w kilku plikach tekstowych na raz!

Kolejną zaletą tego modelu jest niski koszt narzędzi. Część z nich jest darmowa, np. generator stron statycznych i edytor tekstu. Jeśli nawet musimy użyć jakiegoś płatnego oprogramowania, to raczej nie będzie z tym problemu, bo skorzystamy z tego czego używają już programiści w naszej organizacji. W końcu traktujemy naszą dokumentację tak samo jak kod. Dodatkowo, mamy dostęp do tego “co siedzi w środku takich narzędzi”, przez co możemy je poznać na wylot. Dzięki temu będziemy mogli je rozbudowywać i rozwiązywać łatwiej problemy, które napotkamy.

Ale jest też druga strona medalu. Pomimo tego, że pewne narzędzia są darmowe, to ich koszt poniesiemy w innym miejscu - ujawni się on w czasie, który będziemy musieli zainwestować w dodanie brakujących funkcji i w utrzymywanie narzędzia. Na przykład, generator stron statycznych pozwala nam za darmo i dość szybko zbudować ładną stronę z dokumentacją, ale jeśli będziemy chcieli mieć dobrą wyszukiwarkę i generować dokumentację do formatu PDF, to będziemy musieli uzupełnić takie braki we własnym zakresie. Jeśli nasz zespół dokumentacyjny nie ma osób mocno technicznych, to stajemy się skazani na pomoc programistów. A oni mają swoją pracę przy rozwijaniu oprogramowania i zazwyczaj ciężko jest uzyskać ich pomoc. Nie dlatego, że nie chcą tego zrobić, tylko dlatego, że mają na sobie rzeczy o większym priorytecie.

Jak wybrać narzędzie

Wybór narzędzia jest ważny, bo bardzo wpłynie na koszty tworzenia dokumentacji, a

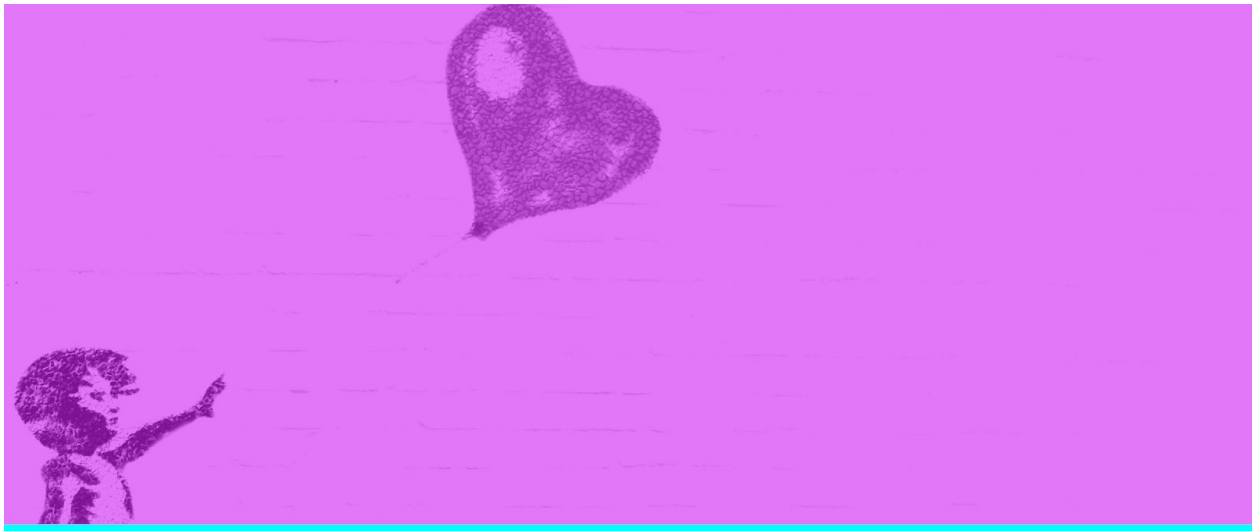
nawet na to jaka ona będzie.

Koszt narzędzia też ma znaczenie. Narzędzie typu Google Docs jest “za darmo”, ale ogranicza nas trochę w kwestii tego jak będziemy publikować treści, co może nas dużo kosztować w perspektywie czasu. Z kolei CCMSy mogą być kosztem rzędu dziesiątek lub setek tysięcy dolarów rocznie, ale mogą się zwrócić w krótkim czasie, i to z nawiązką.

Wszystko zależy od naszych potrzeb. Musimy je dokładnie przeanalizować i porównać z dostępnymi narzędziami. Nie wpadajmy w pułapkę wybierania narzędzia, a potem sprawdzania jak by nam pasowało, tylko dlatego, że inni tego używają!

Nie chcę tutaj rozpisywać się na ten temat, bo można by napisać osobną książkę o procesie wyboru narzędzia. O tym co jest faktycznie potrzebą (publikowanie treści online), a co tylko brzmi jak potrzeba (generowanie release notes z komentarzy z Jirze). O tym jak polityka zakupowa firmy może przedłużyć wybór narzędzia o pięć lat. O tym jak sprzedawcy oprogramowania planują dema tak, żeby nas złapać w pułapkę. Dlatego poprzestańmy na tym krótkim akapicie.

Jeżeli mamy w zespole doświadczonych technical writerów, to pomogą nam skonstruować i przejść proces wyboru. Jeżeli ich nie mamy, a będziemy sporo inwestować w dokumentację, to warto poradzić się konsultanta (niezwiązanego z żadnym narzędziem!)



Posłowie

To koniec tej książki, ale początek Twojej pracy. Mam nadzieję, że moje słowa zainspirowały Cię i jesteś w stanie zaplanować swoje następne kroki. Może zaczynasz pracę nad nowym produktem, a może chcesz poprawić istniejące procesy. Nie zabieram Ci już więcej czasu. Powodzenia!

Podziękowania

Michał Skowron za redakcję, wsparcie, i motywację.

Anna Ligas-Kowaluk za pomoc przy oprawie graficznej.

Tomek Prus, za masę cennych komentarzy.