DEVELOPER NOTES

Tajniki zarządzania zespołem programistów

Łukasz Baran

W artykule zamieściłem zbiór porad adresowanych do kierowników zespołów programistycznych. Jego lektura może być również pomocna programistom, żeby wiedzieli czego oczekiwać od swoich kierowników. W artykule przedstawiam najważniejsze aspekty pracy kierownika zespołu programistów oraz praktyczne porady, które mogą się przydać kierownikom podczas ich bieżących działań. Przedstawione przemyślenia są wynikiem mojej codziennej pracy na stanowisku kierownika, w szczególności prób praktycznego wdrożenia zasad przedstawianych w literaturze.

Rola kierownika w zespole programistycznym

W każdym zespole kierownik ma do odegrania znaczącą rolę. Zajmuje się on organizacją pracy zespołu. Powinien być wsparciem dla członków zespołu zarówno w kontekście posiadanej wiedzy technicznej, jak również w kategoriach socjologicznych. W zespole programistycznym jego rola w znaczącym stopniu pokrywa się z typową rolą kierownika znaną z innych dziedzin. Można też wyróżnić pewien zbiór zadań specyficznych tylko dla kierownika zespołu programistycznego. Przyjrzyjmy się zadaniom kierownika po kolei.

Usuwanie przeszkód

Nie wszystkie przeszkody w realizacji projektu, na które natykają się progamiści, są w stanie samodzielnie usunąć. Czasami potrzebne jest zaangażowanie kierownika. Są to najczęściej problemy natury organizacyjnej, ale mogę również pojawić się trudne problemy technologiczne.

W aspekcie organizacyjnym mogą to być problemy związane z przepływem informacji między innymi zespołami, które współpracują z naszym zespołem. W aspekcie technologicznym mogą to być decyzje związane ze zmianą architektury systemu, zmianą technologii implementacji systemu, użytego frameworku itp.

Czynny udział w tworzeniu architektury systemu

Ze względu na to, że kierownik ma szeroki obraz projektu oraz ma doświadczenia wyniesione z innych projektów, powinien on być angażowany przez zespół architektów w fazie projektowania architektury systemu. Na tym etapie może on być w stanie wykluczyć nierozważne rozwiązania architektoniczne, jak również przedstawić ograniczenia stosowanej technologii, które mogą uniemożliwić realizację przedsięwzięcia programistycznego w wybranym kształcie.

Utrzymanie spójności architektury

Kierownik zespołu programistów powinien być strażnikiem spójności architektury rozwiązania. Powinien pilnować projektu architektury w fazie projektowania, a następnie powinien stać na straży jej realizacji przez swój zespół w fazie implementacji.

Regularne przeglądy kodu

Regularne przeprowadzanie inspekcji kodu prowadzi do podniesienia poziomu inżynierii w zespole, jak również pozwala pilnować spójności architektury rozwiązania. Dodatkowo ogranicza zbiór rozwiązań błędnych oraz możliwość pojawienia się syndrom NIH (ang. Not Invented Here), który jest częstą dolegliwością młodych adeptów programowania.

Mentoring programistów

Kierownik powinien dbać o rozwój członków zespołu. W razie potrzeby powinen też służyć im radą w aspekcie proponowanych rozwiązań implementacyjnych, jak również projektowych. Powinien też pobudzać dyskusję w zespole na temat poprawnych praktyk inżynieryjnych.

Rozdzielanie zadań w zespole uwzględniające kompetencje programistów

Przy rozdzielaniu zadań w zespole kierownik powinien uwzględniać kompetencje poszczególnych pracowników.

Programiści najczęściej specjalizują się w realizacji pewnych elementów systemu. Jeden programista wykona daną pracę szybciej w określonym fragmencie systemu inny wolniej. Kierownik zespołu powinien uwzględniać ten fakt przy podziale zadań.

Niemniej jednak programiści powinny czasem realizować funkcjonalności w modułach, których nie pisali od początku, co pozwala na rozpropagowywanie wiedzy o systemie. Dodatkowo prowadzi do samowolnych inspekcji kodu przez innego programistę, co zachęca często do dyskusji o poprawności przyjętego rozwiązania.

Realizacja zadań zgodnie z harmonogramem

Kierownik powinien dbać, aby zadania były realizowane zgodnie z harmonogramem. Informuje swoich przełożonych o opóźnieniach. Renegocjuje harmonogram z przełożonymi, jeśli nie jest możliwe jego dotrzymanie. Decyduje o poziomie przyjętego „długu technicznego", jeśli zachodzi taka potrzeba.

Implementowanie krytycznej funkcjonalności

Kierownik buduje szkielet architektury implementacyjnej systemu. Może też implementować szczególnie trudne fragmenty systemu, np. wpływające na jego stabilność.

Jeśli zespól posiada dobrych programistów, może im powierzyć te zadania.

Organizowanie zespołu od początku

Ideałem jest gdy zespół działa bez ingerencji kierownika. Kierownik jest niezauważalnym bytem i musi interweniować tylko sporadycznie w wyjątkowych przypadkach. Niestety na początku istnienia zespołu jego rola jest szczególnie ważna ze względu na konieczność kształtowania ducha oraz odpowiedzialności w zespole.

Podejmowanie ostatecznych decyzji

Kierownik powinien być ostatnią instancją, która podejmuje decyzję w sprawach implementacyjnych, niemniej jednak powinien z innymi dyskutować swoje propozycje, gdyż to podnosi ogólny poziom inżynierii w zespole.

Należy też pamiętać, że kierownik nie powinien:

- Blokować przepływu informacji pomiędzy swoim zespołem a zewnętrznym światem. Taka sytuacja jest częsta, gdy kierownicy próbują budować swoją pozycję wprowadzając asymetrię informacji.
- Ograniczać dyskusji w zespole na temat poprawności rozwiązań

Kierownik i sytuacje kryzysowe w projektach

Rozdział ten zawiera zbiór porad co należy robić, gdy w prowadzonym projekcie informatycznym pojawia się sytuacja kryzysowa.

Nowa metodyka niczego nie zmienia

Nagłe wprowadzenie nowej metodyki zarządzania projektami w sytuacji kryzysowej w projekcie niczego nie poprawia.

W projekcie, w którym panuje chaos wprowadzenie żadnej metodyki nie poprawi sytuacji. Najpierw trzeba ustabilizować sytuację. Każda metodyka wymaga nauki, a to odsunie ludzi od projektu na rzecz jej uczenia się i jeszcze pogorszy stan projektu.

Nienegocjowalny harmonogram

Ważne jest tworząc system, aby podzielić wymagania na: „niezbędne", „powinny być", „mogą być". Czasem nie jest możliwe negocjowanie harmonogramu projektu, ale dzięki powyższemu podziałowi zadań można renegocjować zakres projektu. W każdym projekcie najpierw implementuje się wymagania „niezbędne", chociaż mogą być one najtrudniejsze. Nigdy nie zajmujemy się sprawami peryferyjnymi w systemie w początkowych fazach budowania systemu. Należy wykorzystać entuzjazm zespołu na początku projektu do implementacji trudnych przypadków użycia.

Dodatkowy pracownik opóźnia projekt

Wprowadzanie dodatkowych członków do zespołu w sytuacji, gdy mamy problem z dotrzymaniem harmonogramu nie poprawia sytuacji projektu, a wręcz ją pogarsza. Nie ma bowiem liniowej zależności między liczbą osób w projekcie, a czasem realizacji projektu. Jeśli mamy pięć osób w projekcie i jego realizacja ma zająć rok, to dodatkowe pięć osób nie skróci czasu realizacji projektu do pół roku. Wynika to z konieczności uczenia nowych członków (dodatkowo obciąża to starych członków zespołu), niemożności podziału zadań na mniejsze niż to możliwe, aby nowi członkowie byli w pełni obłożeni pracą oraz zwiększenia liczby ścieżek komunikacyjnych w zespole. Co więcej dodatkowi członkowie generują większą liczbę zadań osobom koordynującym ich pracę. Tak więc w fazie projektu po wprowadzeniu nowych członków będziemy mieli często spadek wydajności zespołu, a nie jej wzrost.

Nie załamuj rąk

Projekt zbliża się do końca, a ktoś stwierdza, że cały system nie funkcjonuje jak należy. Najprawdopodobniej nieduża zmiana w systemie, doprowadzi go do działania. Szukaj takiej zmiany, to tzw. punkt dźwigni czyli moment zwrotny w przebiegu wypadków, które mogą przechylić szalę na twoją korzyść. Niemal w każdym projekcie ona istnieje.

Zasada utrzymywania spójności koncepcyjnej całego systemu

W momencie t architekt/programista ma określoną wiedzę i na jej podstawie tworzy pewne wzorce architektoniczne/programistyczne. Po stworzeniu tych wzorców powinien dążyć, aby system był spójny w aspekcie architektury, struktur kodu w całym cyklu życia projektu. Takie podejście ułatwia analizę systemu i pozwala odczytać intencje autora. Łatwiej również taki system przebudowywać w przyszłości.

- Być może w momencie t+1 ten sam architekt/programista uzna, że są lepsze inne wzorce, więc albo powinien przekształcić cały system, aby zachować spójność koncepcyjną albo zostać przy pierwotnych wzorcach.

Pamiętaj o zasadzie Pareto 80/20

Stworzenie 80% funkcjonalności wymaga 20% całkowitego czasu pracy programistów, niestety stworzenie pozostałych 20% wymaga 80%. Prezentuj ten punkt widzenia przy tworzeniu harmonogramów i projektów zaawansowanych, nietypowych funkcjonalności. Te zaawansowane cechy systemu, które najczęściej nie są wykorzystywane przez użytkownika końcowego, doprowadzają do eksplozji kosztów w projekcie.

Nie ulegaj presji na skrócenie harmonogramu lub zmiejszenie zasobów projektu

Kierownik projektu nie powinien ulegać presji zmniejszania harmonogramu projektu.

Dobry kierownik zespołu musi bronić swoich szacunków i nie poddawać się presji wyższego kierownictwa. Na poziomie wyższego kierownictwa często decydują irracjonalne kwestie polityczne. Dodatkowo jeśli kierownik zgodzi się na taki harmonogram, to będzie od niego potem wymagane, aby go zrealizował.

Nie bierz udział w projektach z góry skazanych na niepowodzenie

Najlepiej nie brać udziału w projektach, w których dominuje polityka (firmowa), bo decyzje w takich projektach są często irracjonalne i prowadzą do klęski. Nie poświęcaj dla takiego projektu życia prywatnego, bo są ważniejsze wartości w życiu niż projekt tak uwarunkowany.

Kierownik i podejmowanie decyzji w projektach

Fragment ten zawiera zbiór porad dotyczących podejmowania decyzji. Pokazuje, że nie zawsze udaje się podejmować bezbłędne decyzje w projektach i że jest to stan normalny w procesach podejmowania decyzji.

Niepewność to naturalny stan tworzenia systemu informatycznego

Niepewność jest naturalna i zdarza się nawet doświadczonym kierownikom podejmować błędne decyzje w aspekcie tworzenia kodu czy architektury projektu. Wiele razy kierownik zespołu podejmuje decyzję w warunkach niepewności. Na poziom niepewności w projekcie wpływa nieznajomość technologii, nieznajomość dziedziny problemu. Przy takiej ograniczonej wiedzy często trzeba podejmować decyzję i zdarza się, że czasami będzie ona błędna. Nie podejmuje błędnych decyzji, ten kto ich w ogóle nie podejmuje.

W czasie realizacji projektu niepewność maleje wraz ze zbliżaniem się do końca projektu. Wynika to z poznawania dziedziny rozwiązania, wtedy decyzje powinny być coraz bardziej trafne.

Wyobraź sobie konsekwencje swoich decyzji

Podejmując decyzję spróbuj wyobrazić sobie do jakich konsekwencji ona prowadzi. W aspekcie tworzenia architektury pozwoli to wybrać być może bardziej właściwe rozwiązanie. Kierownik musi w swoim umyśle przeprowadzić pewną symulację działania modelu, architektury.

Monitoruj podjęte decyzje

Każda podjęta decyzja powinna być monitorowana i ewentualnie dostrajana do aktualnych warunków. Monitorowanie decyzji jest bardzo ważne, gdyż wraz z rozwojem projektu wzrasta wiedza na temat jego kształtu i być może aktualnie można zmodyfikować częściowo naszą poprzednią decyzję, aby prowadziła do lepszego rozwiązania. Zmiana decyzji nie jest czymś złym.

Nie da się uwzględniać wszystkich czynników

Nie przejmuj się, gdy w momencie podejmowania decyzji, nie oceniłeś wszystkich możliwości. Często nie jest to możliwe. Tak wygląda naturalistyczny proces podejmowania decyzji.

Ludzie w tym procesie nie podejmują decyzji na podstawie porównywania alternatyw i wybierania najlepszej, a raczej na podstawie poszukiwania analogii do sytuacji, której doświadczyli. Taki model nazywa się modelem podejmowania decyzji opartej na rozpoznaniu. Tak podejmują decyzję eksperci – dowódcy wojskowi, strażacy, szachiści. Wbrew pozorom nie jest to model wykorzystywany tylko przy podejmowaniu decyzji w środowiskach stresogennych. Architekci, programiści również często podejmują decyzje zgodnie z tym modelem.

W tworzeniu oprogramowania często wykorzystujemy również ten model podejmowania decyzji, gdyż wiele decyzji jest podejmowanych ad hoc.

Kierownik a budowanie zespołu

Rozdział ten zawiera zbiór porad dotyczących kształtowania zespołu.

Instruuj programistów jaki jest cel zadania a nie jak go zrealizować

Nikt nie potrafi czytać w myślach. Oprócz wydania polecenia/sformułowania zadania dla programisty, lepiej zamiast mówić jak ma to zrobić (bo zazwyczaj ma kompetencje do zrobienia czegoś), powiedzieć mu jaki jest cel jego zadania. W czym rozwiązane zadanie ma pomóc decydentowi/przedsięwzięciu/operacji. Programista powinien znać kontekst realizacyjny, który ułatwi mu podejmowanie decyzji na poziomie implementacyjnym.

Model podejmowania decyzji opartej na rozpoznaniu

1. Oceniamy czy sytuacja jest analogiczna do przeszłej lub czy jest nowa
   - jeśli analogiczna → przystępujemy do działań
   - jeśli nowa → przechodzimy do punktu 2
2. Jeśli mamy nową sytuację staramy się przeprowadzić symulacje jej przebiegu. Następnie rozpoczynamy działanie zgodnie z przeprowadzoną symulacją.
3. Oceniamy cały czas działanie i prowadzimy symulację myślową jej dalszego przebiegu. Jeśli coś nie idzie zgodnie z planem, przeprowadzamy symulację i modyfikujemy działanie.
4. Wszystko idzie zgodnie z planem, to doprowadzamy działanie do końca.

Nie psuj samoorganizujących się zespołów

Nie psuj samoorganizującego się zespołu. Pozwól programistom – pasjonatom pracować. Nie trzeba kontrolować każdego ich kroku. Nie ograniczaj inicjatywy, pozwól działać wolnym elektronom, tylko kieruj ich zapał tam gdzie jest potrzebny.

Organizuj tak miejsce pracy, aby ludzie mogli utrzymywać koncentrację

Miejsce pracy programisty powinno być miejscem spokojnym, gdzie można skupić myśli. Ogranicz telefony, wstaw drzwi izolujące od całego zgiełku firmowego. Open space'y są złe dla wydajności zespołów programistycznych. Postaraj się zorganizować tak przestrzeń, aby unikać nawet kilkusekundowych wytrąceń z koncentracji takich jak zbędne telefony i maile.

Dodatkowo w jednym przedziale czasowym programista powinien wykonywać jednocześnie tylko pojedyncze zadanie i nie powinien przełączać się pomiędzy zadaniami. To znacząco poprawia jego wydajność.

Programowanie to praca wymagająca głębokiej koncentracji dlatego niezbędne się staje odizolowanie ludzi uczestniczących w projektach od spraw bieżących. Jeśli bieżących spraw jest dużo (np. związanych z utrzymaniem systemu) to najlepiej wyznaczyć jednego programistę, który będzie dyżurował i rozwiązywał je w trybie ciągłym. Pozostali w tym czasie będą mogli się skoncentrować na zadaniach w projekcie. Pozycja dyżurnego programisty musi być rotacyjna.

Pieniądze nie zawsze są dobrym motywatorem

Dla młodszych programistów dobrym motywatorem są pieniądze, bo zarabiają względnie mniej niż starsi koledzy. Ale dla starszych programistów pieniądze najczęściej nie są motywatorem, bo od pewnego poziomu nie da się motywować pieniędzmi. Z pieniędzmi jest jak z higieną ich brak powoduje niezadowolenie, ale jak już masz pewną ilość, to przestajesz je doceniać. Starszym programistom po długim projekcie można zorganizować szkolenia, dać możliwość poznania nowej technologii, pozwolić na dłuższe wakacje tzw. Projekt X.

Każdy może mieć gorszy okres w pracy

Dzień, tydzień, miesiąc nie udaje ci się skoncentrować na pracy. Klikasz bez sensu pocztę, odwiedzasz bezsensowne strony, nie przejmuj się każdego dopada czasami taki stan, on przemija. Aby to przyspieszyć po prostu „Zacznij pisać kod, który miałeś napisać". Można to też skrótowo ująć „Wystarczy zacząć" lub „Ognia i naprzód!".

Oceniaj pracę nigdy człowieka

Pamiętaj w czasie robienia przeglądów kodu, że oceniamy kod nie człowieka i tak powinien to również rozumieć każdy członek zespołu.

Deleguj szeroką odpowiedzialność za zadanie

Zorganizuj tak zespół, aby móc powierzać programistom szeroką odpowiedzialność za zadanie. Delegowanie zadań pozwala odciążyć kierownika zespołu od spraw bieżących, dodatkowo najczęściej poprawia przepływ informacji pomiędzy twoim zespołem a światem zewnętrznym.

Pozwól na swobodę przepływu myśli, dyskusji

Pytamy, pytamy, pytamy... Nie ma głupich pytań. Nie poddajemy się presji innych, że nas wyśmieją, bo czegoś nie rozumiemy. Taki swobodny przepływ myśli prowadzi do podniesienia poziomu inżynierii w zespole.

Kierownik a organizacja i technologia

Ten fragment artykułu przedstawia porady jaką postawę przyjąć w stosunku do pewnych uwarunkowaniach organizacyjnych oraz technologicznych, w których pracujemy.

Organizuj i zarządzaj wiedzą w firmie

Wiedza jest zasobem dla firmy, który należy pielęgnować. Organizuj wiedzę w firmie, nie pozwól żeby pozostawała ona tylko w umysłach ludzi. Wiedza powinna zostawać w organizacji w jakiś trwały sposób. Najlepiej, żeby to był elektroniczny łatwozarządzalny magazyn wiedzy, jak na przykład wiki, jira.

Nigdy nie wyrzucaj starego kodu do kosza

Nie pisz od początku już gotowych funkcjonalności, które istniały tylko dlatego, że kod jest przestarzały. Wystarczy przeanalizować przypadek Netscape'a, który przez taką decyzję upadł. Wyrzucił najbardziej popularną wyszukiwarkę do kosza i zaczął pracę nad nową jej wersją od początku, w tym czasie Microsoft zdominował rynek przeglądarek ze swoim niedopracowanym Internet Explorer'em. Programiści wolą przepisać kod, bo czytanie kodu jest trudniejsze od jego pisania, dlatego obalaj takie pomysły programistów. W niektórych przypadkach oczywiście przepisanie jakiejś funkcjonalności może mieć sens.

Harmonogram jest nieodłącznym elementem każdego projektu komercyjnego

Naucz się żyć z harmonogramami, one muszą istnieć, gdyż przedsięwzięcia informatyczne są komercyjne, a więc muszą przynosić dochód. Niemniej jednak harmonogram nie może psuć jakości systemu. Ty jako kierownik powinieneś ustalać poziom „długu technicznego" w momencie, gdy kierownicy wyższego szczebla, chcą wywrzeć na Ciebie presję przyspieszenia prac w projekcie.

Używaj technologii dojrzałych i pasujących do Twojego projektu

Wybory technologii powinny być podyktowane rzeczywistymi potrzebami projektu, a nie potrzebami osób uczestniczących w projekcie. Programiści często forsuje „metodykę" resume driven development, która umożliwia im wpisanie nowych technologii do swojego cv. Nie zgadzaj się na takie postępowanie. Dbaj o interes projektu.

Testowanie może być dłuższe niż implementacja

Uwzględniaj fazę testowania i usuwania błędów w projekcie, czasem może to pochłonąć nawet 50% całkowitego czasu realizacji projektu. Często trzeba to uzmysłowić osobom zarządzającym projektem.

Utrzymuj aktualność specyfikacji funkcjonalnej

Specyfikacje funkcjonalne są ważne, gdyż uzmysławiają programiście jak cały system działa i powinny być aktualizowane aż do zakończenie implementacji, gdyż wtedy pojawia się w niej najwięcej zmian. Uszczegóławianie rozwiązania na etapie implementowania systemu powoduje znalezienie wielu niedociągnięć, czy braków w specyfikacji, które powinny być uwzględniane,

jeśli posiadanie specyfikacji funkcjonalnej w danym projekcie ma mieć sens.

Testuj ergonomię swojego rozwiązania

Spróbuj za pomocą systemu, który przygotowujesz zrobić to do czego jest przeznaczony, wtedy sam stwierdzisz jakie jeszcze posiada braki. Zanim klient uzyska tak wybrakowany produkt, będziesz mieć jeszcze czas na poprawki funkcjonalne.

Pamiętaj o prawie nieszczelnych abstrakcji i poznawaj nieustannie środowisko pracy

Pomimo, że znasz się na jednej technologii (np. Java Enterprise) czasem musisz zagłębić się w inne obszary. Należy nieustannie rozwijać swoje umiejętności wybiegające często poza nasze obecne kompetencje. Wynika to z tego, że często trzeba rozwiązać problem, który wykracza poza podstawowy obszar naszej działalności, a zmusza nas do tego właśnie prawo nieszczelnych abstrakcji.

System musi się kompilować i uruchamiać od samego początku jego życia

System musi być cały czas kompilowalny i uruchamialny. Uruchomienie środowiska dla naszego systemu nie powinno sprawiać trudności niedoświadczonemu programiście. Najlepiej, żeby nasz system budował pojedynczy skrypt.

Podsumowanie

W artykule zawarłem zbiór porad, który ma pomóc organizować pracę kierownikom zespołów programistycznych. W szczególności zwracam uwagę na to, jak tworzyć warunki sprzyjające rzetelnej pracy.

Poniżej przedstawiam literaturę, z którą warto się zapoznać, szczególnie w momencie, gdy rozpoczynamy pracę jako kierownik zespołu programistycznego. Znajduje się w niej bardzo wiele cennych wskazówek, jak radzić sobie z niektórymi sytuacjami związanymi z zarządzaniem zespołem informatycznym lub projektem informatycznym. Co więcej pozwala ona spojrzeć na zarządzanie zespołem informatycznym z szerszej perspektywy i z pewnym dystansem.

Literatura

1. T. Demarco „Czynnik ludzki. Skuteczne przedsięwzięcia i wydajne zespoły", WNT
2. R. Whitehead „Poradnik kierownika zespołu", WNT
3. F. Brooks „Mityczny osobomiesiąc. Eseje o inżynierii oprogramowania", WNT
4. E. Yourdon „Marsz ku klęsce", WNT
5. J. Spolsky „Zarządzanie projektami informatycznymi", Helion
6. G. Klein „Sztuka podejmowania decyzji", Helion

O AUTORZE

Łukasz Baran, Inżynier Systemów JEE, Kierownik Zespołu

Inżynier oprogramowania w e-point SA, gdzie zajmuje się projektowaniem i implementacją systemów internetowych w technologii JEE. Absolwent Wydziału Elektroniki i Technik Informacyjnych Politechniki Warszawskiej oraz Szkoły Głównej Handlowej. Interesuje się platformą Java SE, Java EE oraz dobrymi praktykami programistycznymi.

Kontakt: **lukasz.baran@e-point.pl**