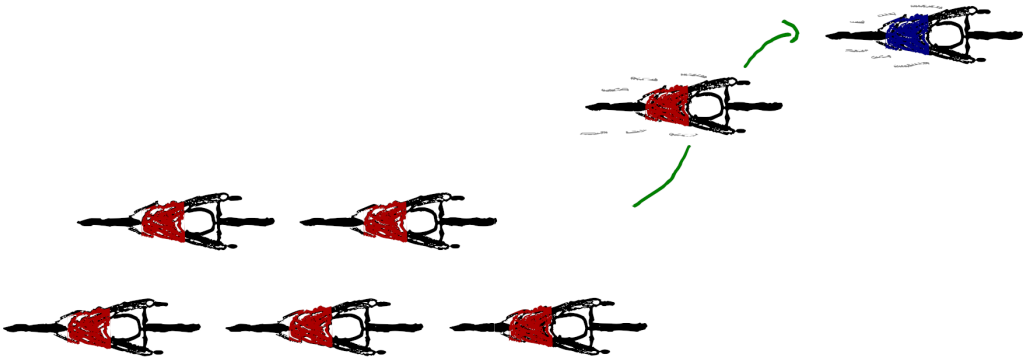


Engineering Team Lessons from Cycling

Posted October 15, 2023 under Leadership, XP.

Tweet

Cycling provides interesting examples for software development. It's possible to race individually or in teams. A group that's an effective team will outperform the same group acting as individuals every time. Teams compete towards a goal, and also against the environment, and many other teams, all with their own tactics, all at the same time.

Ensemble working provides an Advantage

Cycling teams working together in a paceline can travel significantly faster—via the aerodynamic advantage of drafting the slipstream of the rotating leading rider.

More Articles

Engineering Team Lessons from Cycling

One does not simply deliver software

Do you need a Strong Leader?

Supporting Sustainability

Pondering Agile Principles

Cost of Attrition

Uncovering Better Ways

Don't hire top talent; hire for weaknesses.

Escape the Permission Trap with Healthy Habits

Thinking in Questions with SQL

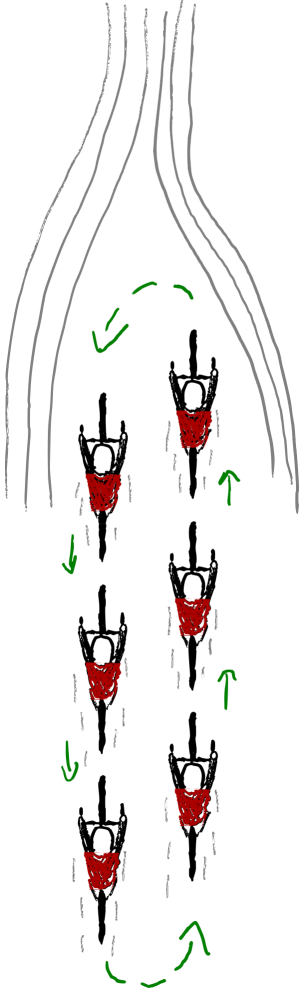
Leadership Language Lessons from Star Trek

Java 16 Pattern Matching Fun

We got lucky

Revisiting Html in Java

Meetings, ugh! Let's change



Software teams can build more stuff if each person works independently, than when working together. However, they'll reach a team goal the fastest through working together, on the same stuff, at the same time, via pair or ensemble programming.

Having all the perspectives, experience, and expertise available at the point of meeting resistance from unknown challenges, results in a sort of aerodynamic advantage. We get stuck for less long, and overcome obstacles more easily.

A pair may not be twice as fast at shipping stuff as two people, but shipping stuff is not important. Working together they'll achieve a goal a bit faster. Our goals have a cost of delay. The ability to ship a capability with associated revenue significantly faster, more than makes collaborative working worthwhile.

our language

Latency Numbers Every Team Should Know

Humility

Sealed Java State Machines

A little rant about talent

Fun with Java Records

The benefits of making code worse

Reasons to hire inexperienced engineers

Do you CI?

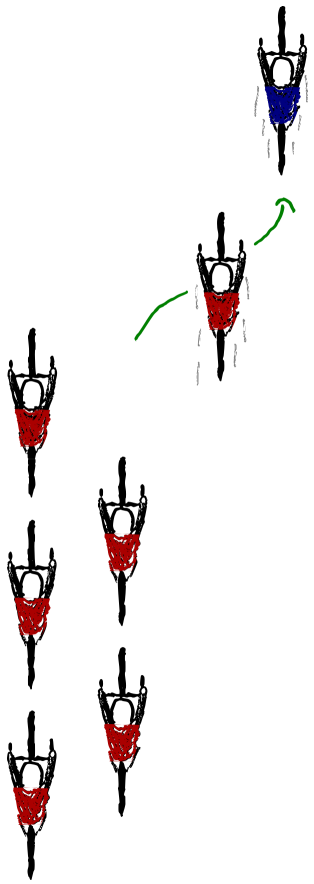
Learning from Pain

The unsung upsides of staying put

Follow @benjiweber

Individual working provides an Advantage

Cycling teams are comprised of multiple people, they can send riders back for hydration & energy food. They can send riders ahead to mark breakaways and mitigate the risk of the breakaway succeeding at remaining ahead until the end of the race.



Software teams that are paying attention, will see side opportunities & risks that are tangential to the main focus but too big to ignore. Often it helps for one or two people to go after them. Builds getting slow; someone peels off from the group to go fix that. Competitor launched a new product; someone peels off to explore how it works and whether should this change our plans.

Competitive Market

Unlike many pitch team sports where teams compete head to head, cycling teams compete against several other

teams, at the same time. Each competitive team will employ their own tactics. Some teams may be more interested in overall win, or other incentives available such as intermediate sprints. Tactics have to take into account all the competition, not just a single opposing team. There are also environmental factors on the day such as wind and rain conditions, and the terrain.

Organisations building software are competing against many other organisations with directly or indirectly competing interests.

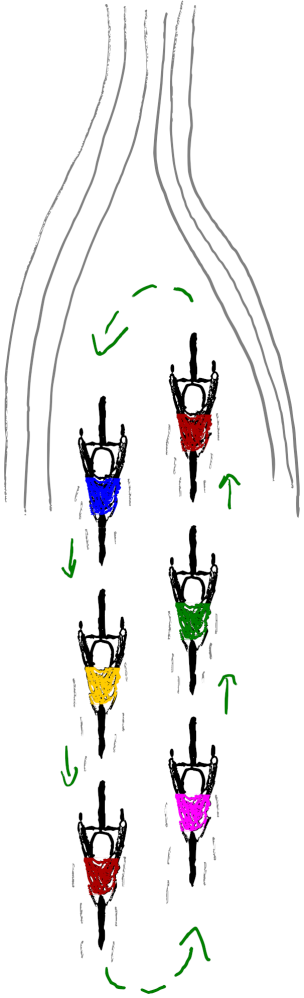
Sometimes it is advantageous to collaborate with the competition to conserve resources by working together. Cycling teams can work together to increase the chance of leading the race through a breakaway. Software builders partnering with competitors can build an ecosystem to grow the opportunity available to all parties.

Other times there's an opportunity to get ahead of the competition by creating a break that your team is uniquely positioned to take advantage of.

Adapt to the riding conditions. Lightweight for climbing or Aero for flat. Wet or Dry. Similarly, as the economic climate changes, so do the tactics that are most successful for building software.

Dynamic Reteaming to Breakaway

Working together, a breakaway made up of individuals from multiple teams can stay ahead of the race. They must take their own share of the hard work in the wind, and communicate effectively. If they refuse to help each other, the breakaway will fail, and will be absorbed. Breakaways often fail when competing incentives and ineffective communication reduce their advantage over the main peloton.



There’s often a need and benefit to pulling together or self organising temporary short lived software teams. They can effectively chase an immediate goal that doesn’t neatly map to the existing team structure.

Dynamic reteaming can be far more effective than dependencies between teams, and like a breakaway it can rapidly get ahead of what an unchanged organisation structure could achieve.

Temporary teams also often struggle to communicate as effectively as a well established team; lacking the trust that comes from experience working with each other.

Marginal Gains Add Up

A clean well-lubricated chain could save you 5 watts, adding up to several seconds advantage over a race. Small tweaks to fitness training habits every day can add up to an advantage over a season.

Software teams often underestimate how quickly small investments in their own effectiveness cumulatively create an advantage. Time spent making your deploy pipeline a couple of minutes faster, or time automating manual onboarding steps. The wins from these add up over time to help you outpace the rest.

Specialists

You need specialists: climbers & sprinters. The rest of the team help get them in position at the right time for the challenges that they can best help with. Get your sprinters to the front of the race with a leadout in time for the sprints. Do the hard work to protect your race-winner's ability to take advantage of a tactical opportunity when it comes. However, everyone needs to finish the race. Sprinters must be able to get over the mountains within the time limit.

Effective software teams have a mix of generalising specialists. They can all muck-in with whatever needs to be done towards the team goal. There's also huge value to having the person with expertise in building UIs, the person with Data Science expertise, or the person who's scaled databases. The right specialists enable help the team to overcome challenges.

Support Staff

Teams need support staff. If riders had to stop and repair their own mechanical issues, or carry their own food they'd be far slower.

Engineering teams benefit from internal platforms, and developer experience tooling support functions. Invest a portion of your budget in these.

Beware Incentives

Unglamourous work is essential to team success. Teams are ruined if everyone tries to go for the win.

Cycling teams need domestiques who will protect the leaders and fetch food. Software teams need people who'll do glue work. People who help the whole team communicate. People who remove the friction slowing the team down.

Organisational incentives often risk this essential work and destroy the potential of teams.

Incentivise each of your riders to go for the win and the team will fail.

Incentivise each of your developers to chase promotions and compensation increases, on the basis of their individual impact, and your team will be ineffective.

> Incentivise each of your developers to chase promotions and compensation increases, on the basis of their individual

> impact, and your team will be ineffective.

Work Sustainably

Pace yourselves. Only go all out when there's a need for it, or you'll not have the energy when it is needed. There's a sprint at the end of the race. There's another stage tomorrow. Don't drop out.

If you are successful, there will be production incidents and customer crises to handle. These will take your energy reserves.

If you run at 100% every day you'll have nothing left to give when there's a real need.

Supporting Sustainability

Many software teams struggle with ever growing cost of change, and technical debt that risks overwhelming them. It's always interesting to talk with folks and understand, how the system and incentives created this state. Often I hear from software engineers that management doesn't give them permission for (or doesn't prioritise) work such as tidying, refactoring, … Continue reading

Benji's Blog 0

Leave a Reply

Name (required)

Your Name

Mail (required)

Your Email

Website

Your Website

Your Comment Here…

Submit Comment

Admin

- Log in
- Entries feed
- Comments feed
- WordPress.org