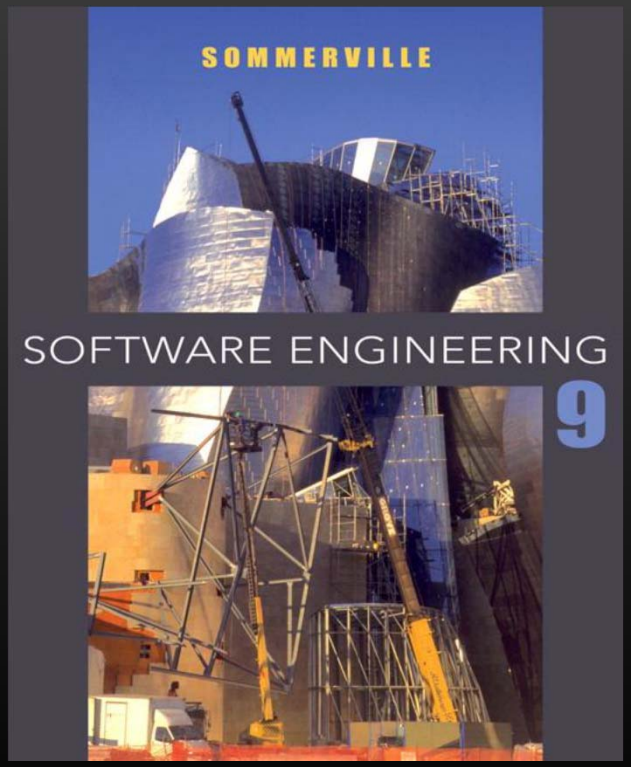
Analyzing Engineering Process Data at Microsoft: *What's the Opportunity?*

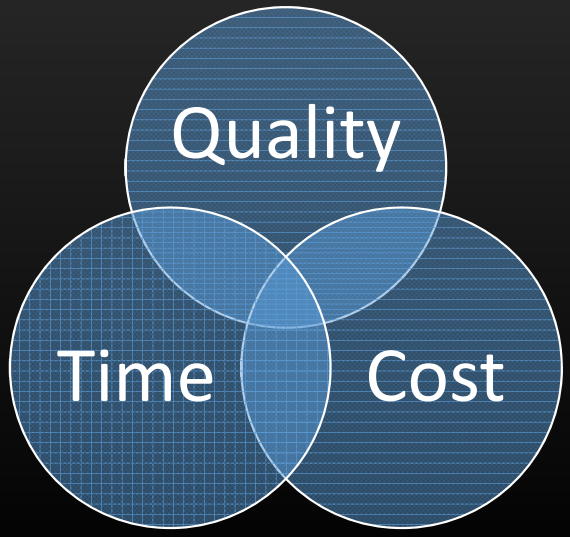
Wolfram Schulte
Principal Researcher and Partner Development Manager
Microsoft Corporation

26th Annual IEEE Software Technology Conference 2014
1 April 2014 | Long Beach, California, USA

Software Engineering (SE)



"Produce high quality software with a finite amount of resources to a predicted schedule"



SE Information Needs …

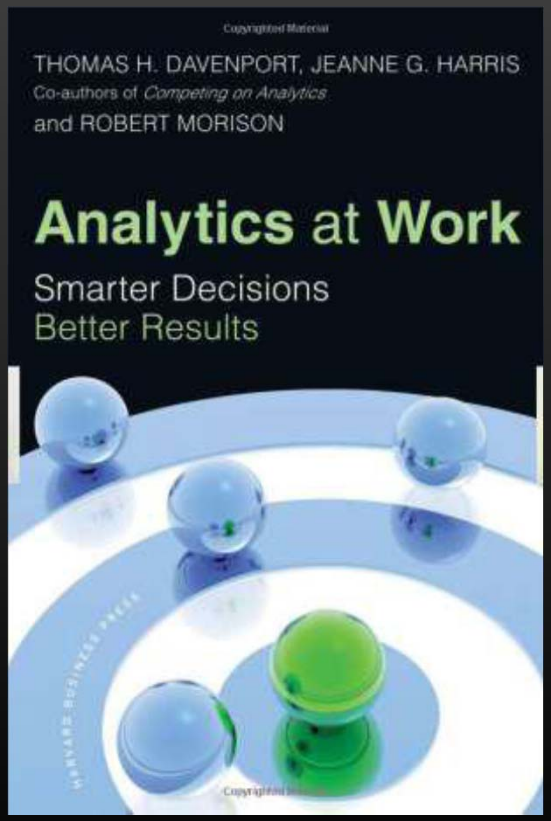
Engineers want to know

- what's going on in **code** (churn)
- and what **dependencies** exist
- where are the **bugs**
- who are the **people** involved

Managers want to know

- how to set-up an **effective dev. process** and **organization**
- what's the **quality of the product**
- can we ship **on time**
- what's the **customer experience**

SE Analytics



"Use of [SE] data,
analysis and systematic reasoning to
[inform and] make decisions"

Use of Analytics

	Past	Present	Future
Infor-mation	What happened? ***Reporting***	What is happening now? ***Alerting***	What will happen? ***Extrapolation***
Insight	How and why did it happen ***Modeling***	What's the best next action? ***Recommendation***	What's the best/worst that can happen? ***Prediction***

From Davenport et al. "Analytics at Work".

Optimizing for Time …
Branch Analytics

Branching in Source Control Management Systems

Coordinating the work of 100's of developers is challenging.
A common solution is to **use branches in SCM** systems

- Benefits: Isolating concurrent work during times of instability
- Cost: Increase the time that changes percolate through the system (Code Velocity)

Branches, Flow, Velocity, and Volume

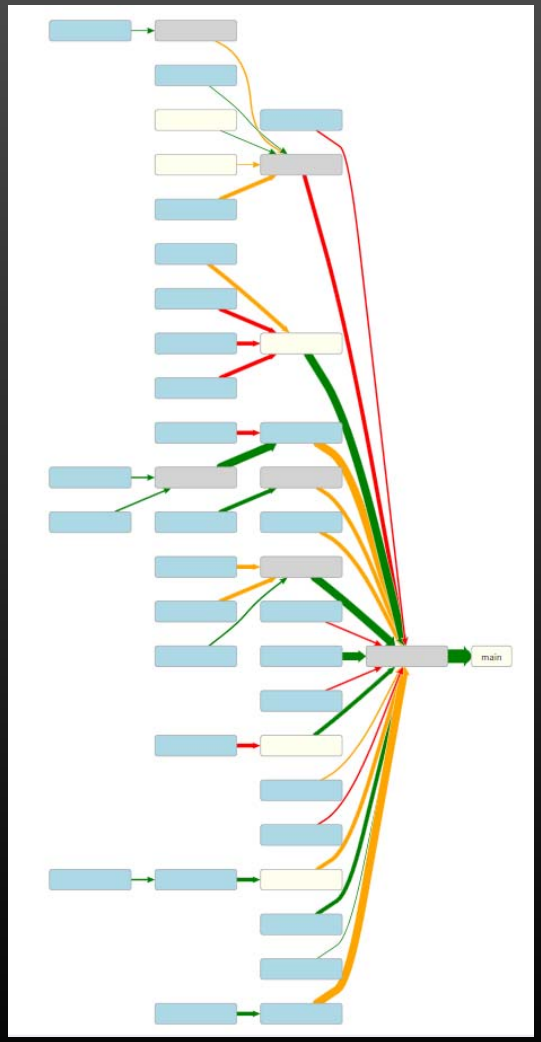
Nodes: Branches

[Size: # of developers]

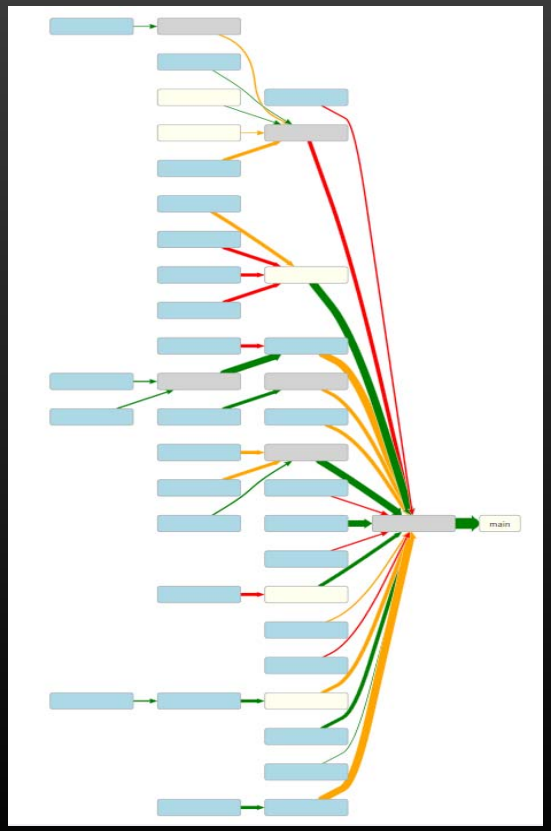
Edges: Flow of Code

Thickness: Volume

Color: Speed

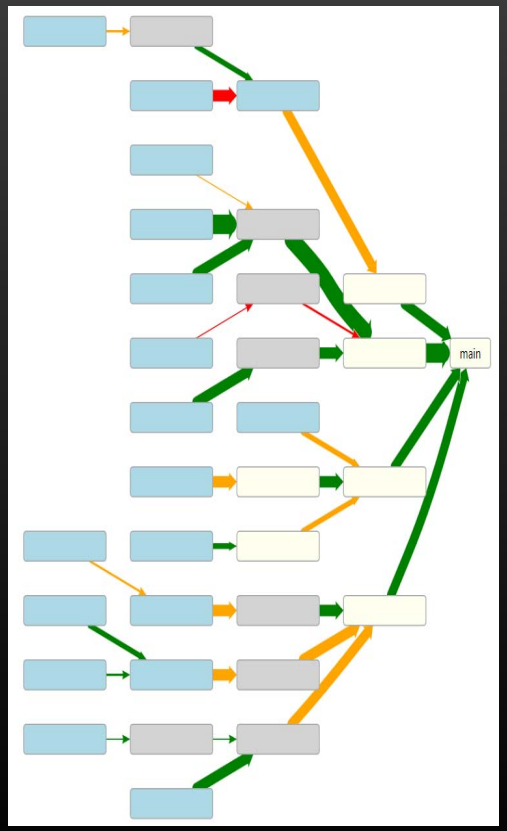


Branch Structure Optimization



Identify branches that contribute to delay and restructure

?

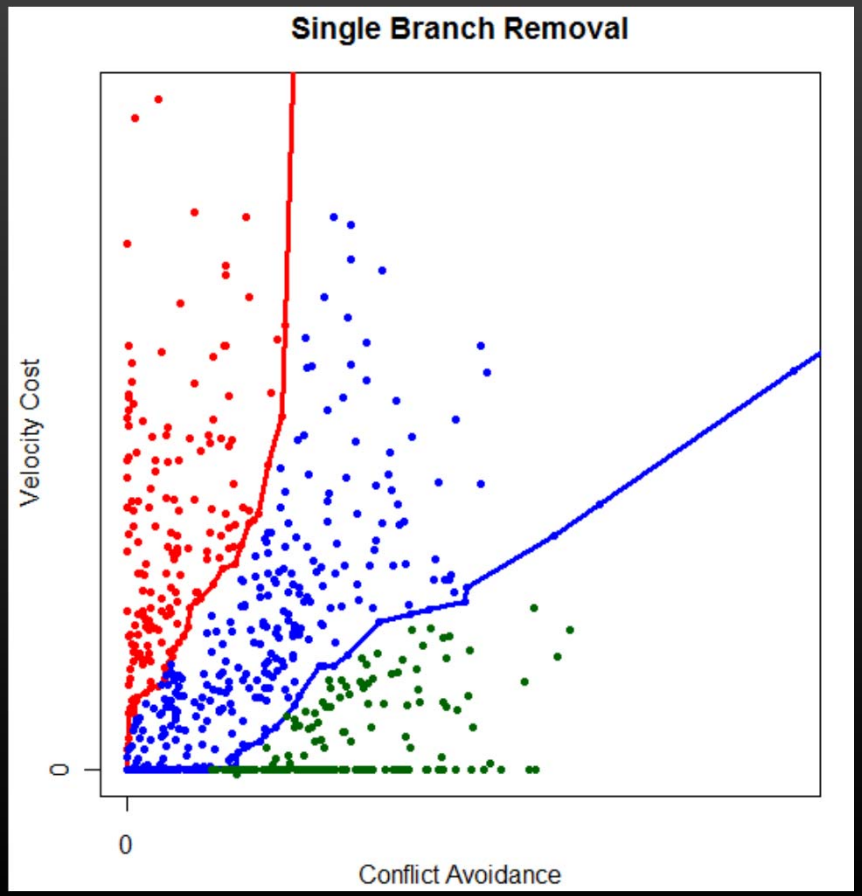


Simulate Cost-Benefit of Alternative Branch Structures

Idea: Replay code churn history with each feature-branch removed

Measure impact on:

- Velocity (“cost”)
- Avoided conflicts (“benefit”)



Results

- Faster velocity: ~35% more code reaching main trunk per day
- Better reliability: ~50% fewer merge conflicts
- Infrastructure savings ($): Having fewer branches to build and maintain

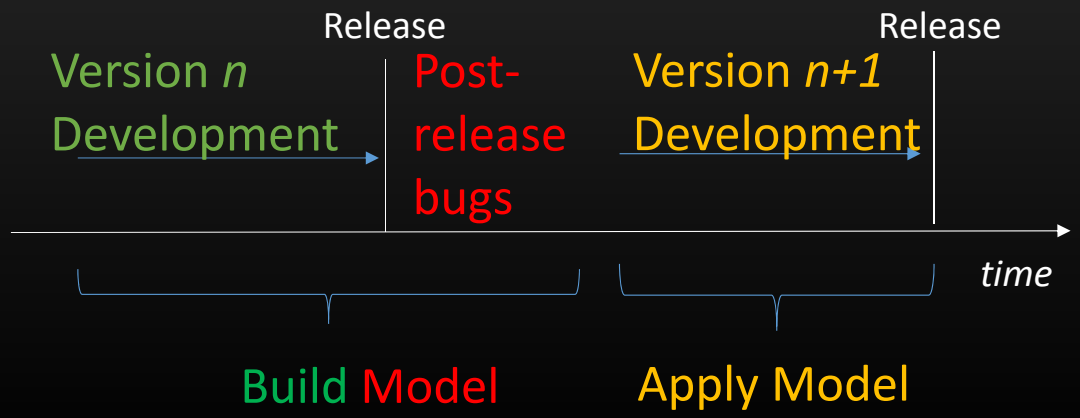
Assessing Quality
Risk analysis

Risk Prediction

Risky change: something likely to cause fixes in the future

Risk prediction: evaluating risk of each pre-release check-in using a statistical model

Approach: Use post-release data to decide on risk for pre-release changes



Predictors & Model

Metrics for check-ins are computed at the file level, then aggregated for check-ins

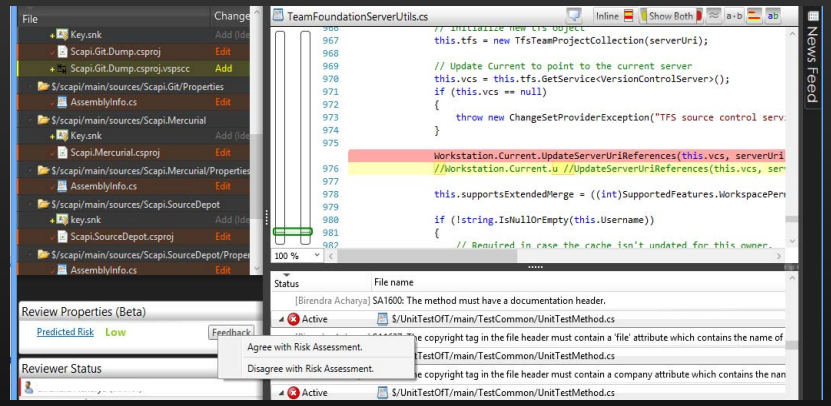
- Code metrics
- Code churn (present and past)
- Bugs
- Ownership
- Developer experience

Use logistic regression model

Probability of risk is linearly proportional to values of metrics

Risk Analysis Usage

Code Review Tool for Engineers



Dashboards for Managers

Checkin filtering criteria

Change made within: 2 days

depot name: All

changeID:

branch name: All

file name contains:

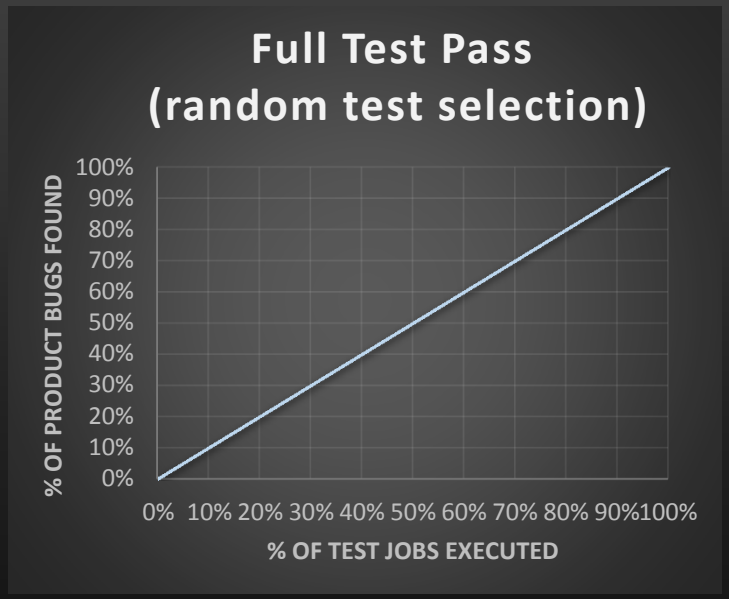
Apply Filter

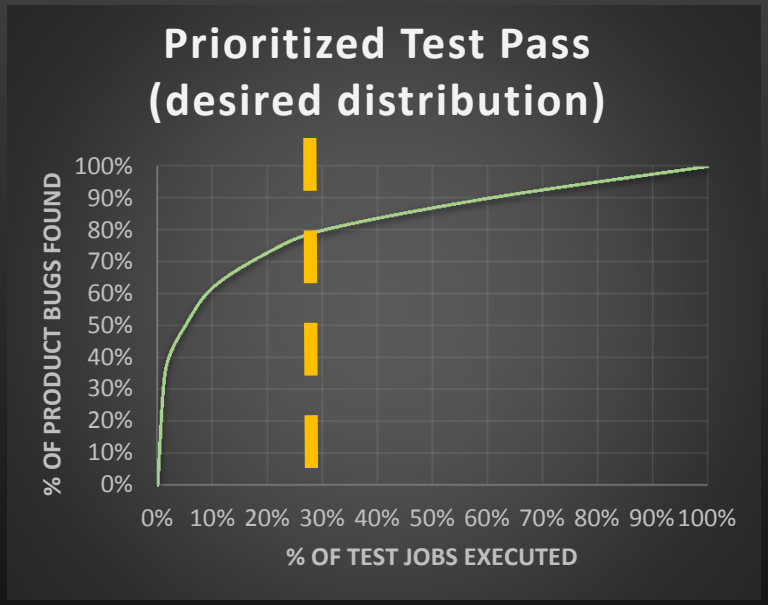
Depot name	Changelist ID	Change date	Changed by	Files added	Files changed	Files deleted	Files integrated	Risk
	2688659	7/16/2013 6:42:10 PM		0	3	0	0	0.4127268
	2688664	7/16/2013 6:42:46 PM		1	13	0	0	0.6551605
	184514	7/16/2013 6:44:42 PM		0	14	35	1	0.4456684
	2688669	7/16/2013 6:48:59 PM		0	1	0	0	0.1768458
	184515	7/16/2013 6:44:54 PM		0	1	0	0	0.1768458

Allows for smarter decisions, where to focus, i.e. how to use time and resources

Trading Time For Quality?
Test Priorization

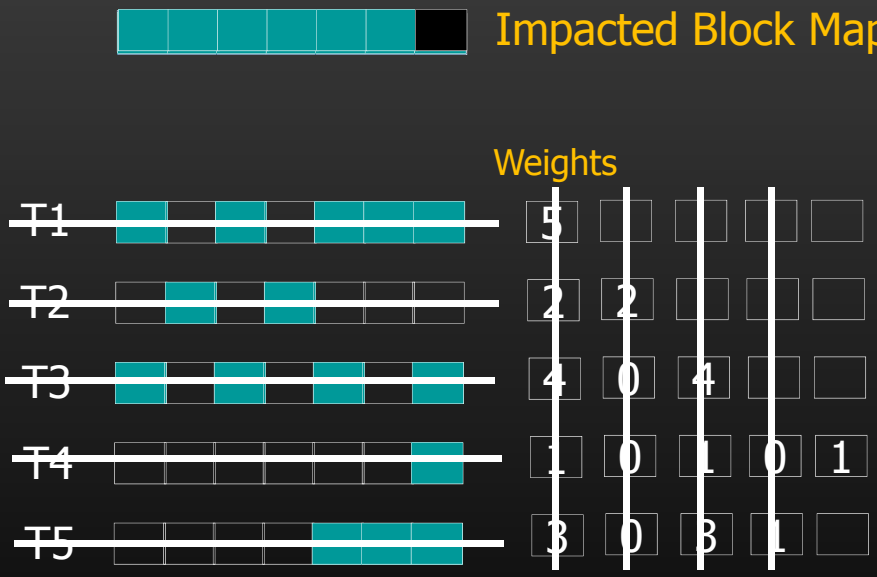
Test Prioritization Concept





- **Goal**: for bugs that are discoverable with existing tests, find them early in the test pass
- Use information about changes to order tests to max. chances of achieving the goal
- Draw **a line** though a prioritized test suite → **test selection**

Basic Test Priorization Idea



Covering Set 1

T1

T2

Covering Set 2

T3

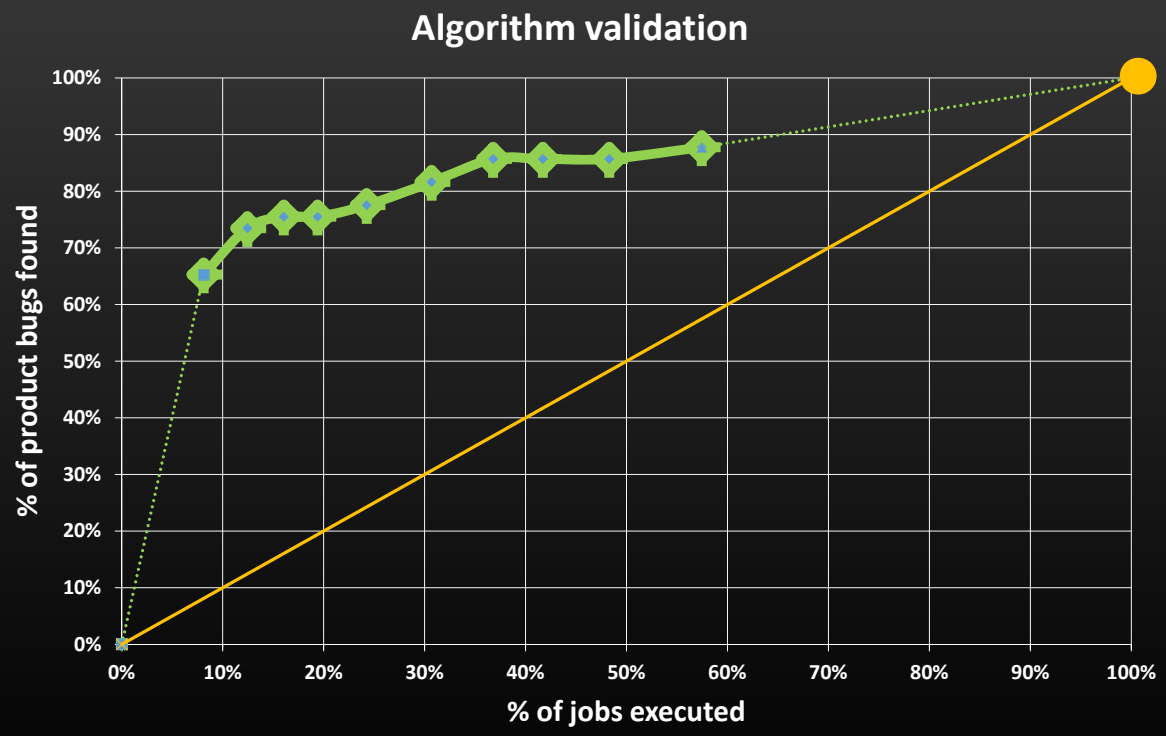
T5

Covering Set 3

T4

Denotes that a trace T covers the impacted block

Effectiveness



Comparing:
3 month SxS experiments

- 800k+ tests, ~380 hours of total machine time with
- 13,800k+ tests and 4,500 hours of total machine time

Results

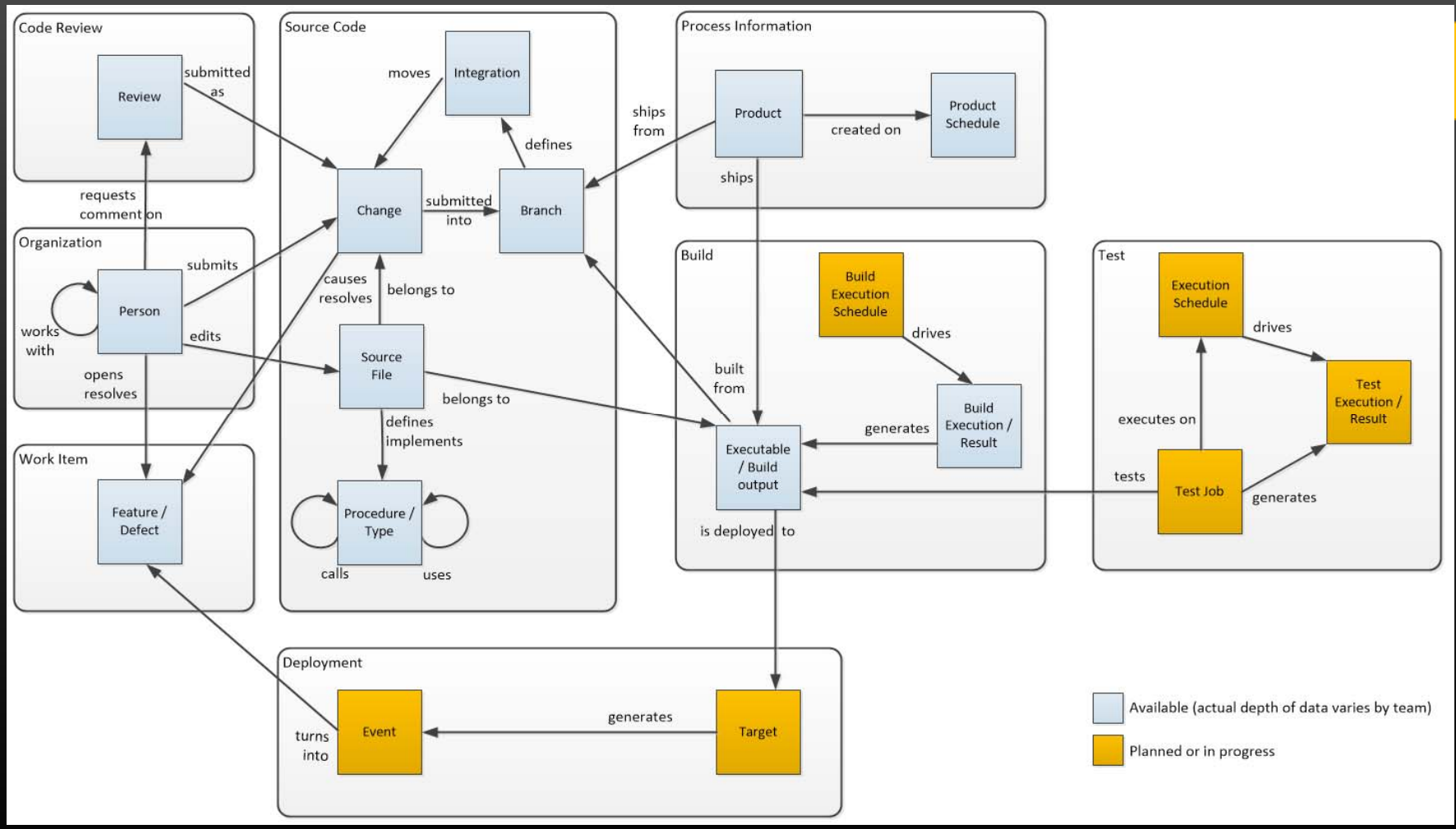
- Faster daily tests with good effectivity
  - Running only 20% of tests daily with 78% bug detection rate
- Same reliability
  - Running all tests on the weekend with 100% bug detection rate
- Infrastructure savings ($)
  - Having fewer test machines to maintain

The Enabler: CodeMine

Collects all development process data (people, sources, bugs/features, code reviews) for all large product groups, makes it queryable and provide analysis on top.

Currently mining ~100 repositories; ~30TBs of data, 200 direct active users + tools + dashboards

Depth of CodeMine



CodeMine-enabled Scenarios :: Examples

Enable **company wide special purpose tools** (in 7 BGs)

- Branch optimization
- Risk analysis
- Test priorization

Drives **product specific dashboards** (in 4 BGs)

- Churn, risks, bugs, ownership, velocity, build & test verdicts,…

Allow **custom queries for one-off engineering decisions** (in 6 BGs)

- Onboarding, build break analysis & alerting, perf. analysis, Org optimization, …

Lessons learned

- Uniform representation
- Individual instances
- Policies for security, etc
- Encode process information
- Federate data access
- Discoverability

Engineering data is a gold mine for process, practice, tool improvement

http://research.microsoft.com/~schulte

For papers please
search for the websites of
the people on the right
e.g. enter “Chris Bird Microsoft”

or simply:

http://research.microsoft.com/ese/
http://research.microsoft.com/tse/

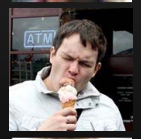
Insights by

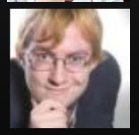

Chris Bird


Jacek Cerwonka

Brendan Murphy


Nachi Nagappan


Alex Teterev (no website)


Tom Zimmermann